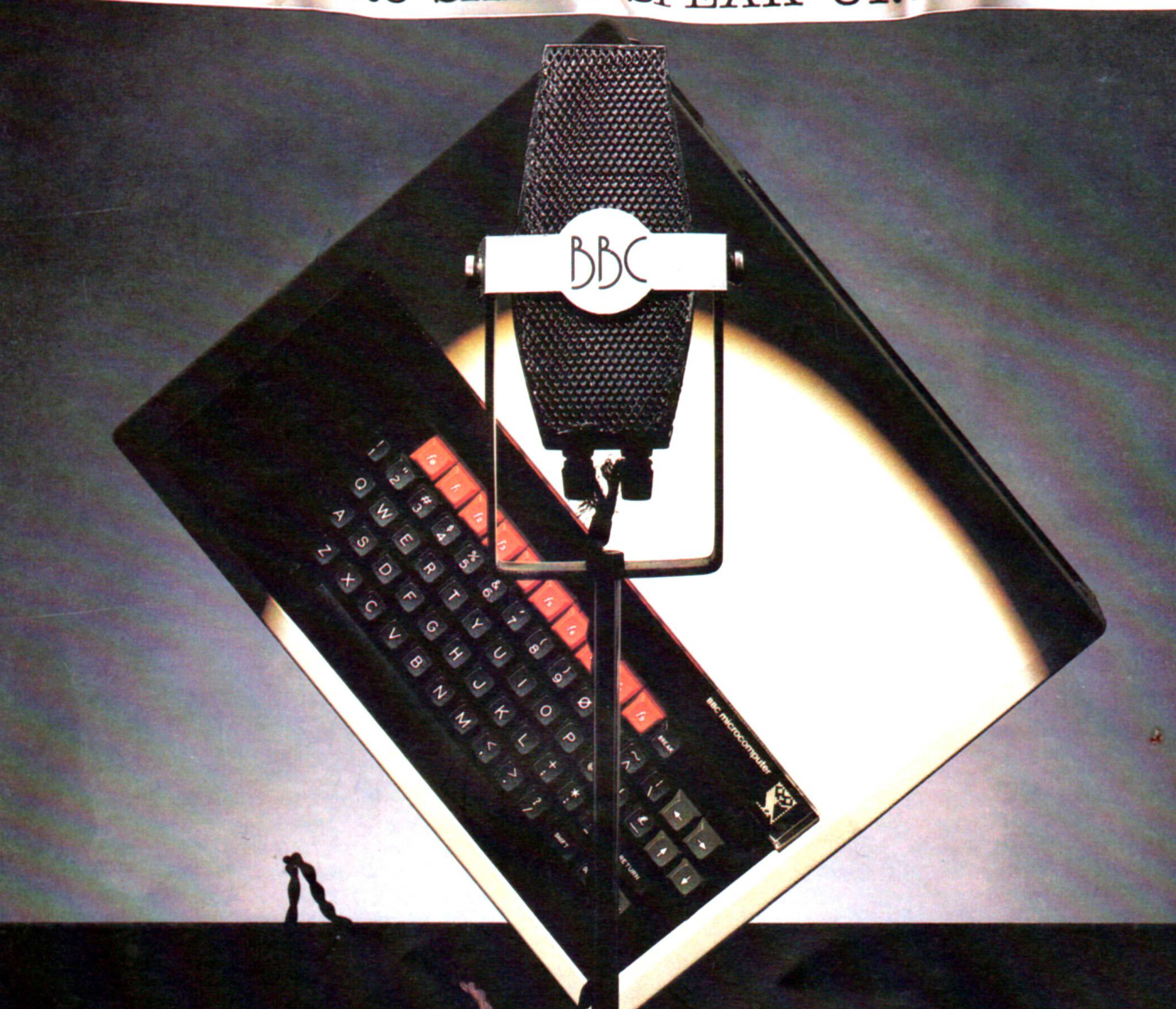
37

mi computer

CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

AND MICRO SHALL SPEAK UNTO MICRO

150 ptas



Editorial Delta, S.A.

MI COMPUTER CURSO PRACTICO DEL ORDENADOR PERSONAL, EL MICRO Y EL MINIORDENADOR

Publicado por Editorial Delta, S.A., Barcelona

Volumen IV - Fascículo 37

Director: José Mas Godayol
Director editorial: Gerardo Romero
Jefe de redacción: Pablo Parra
Coordinación editorial: Jaime Mardones, Francisco Martín
Asesor técnico: Ramón Cervelló

Redactores y colaboradores: G. Jefferson, R. Ford, F. Martín, S. Tarditti, A. Cuevas, F. Blasco
Para la edición inglesa: R. Pawson (editor), D. Tebbutt (consultant editor), C. Cooper (executive editor), D. Whelan (art editor), Bunch Partworks Ltd. (proyecto y realización)

Realización gráfica: Luis F. Balaguer

Redacción y administración:
Paseo de Gracia, 88, 5.°, 08008 Barcelona
Tels. (93) 215 10 32 / (93) 215 10 50 - Télex 97848 EDLTE

MI COMPUTER, *Curso práctico del ordenador personal, el micro y el miniordenador*, se publica en forma de 96 fascículos de aparición semanal, encuadernables en ocho volúmenes. Cada fascículo consta de 20 páginas interiores y sus correspondientes cubiertas. Con el fascículo que completa cada uno de los volúmenes, se ponen a la venta las tapas para su encuadernación.
El editor se reserva el derecho de modificar el precio de venta del fascículo en el transcurso de la obra, si las circunstancias del mercado así lo exigieran.

ISBN: 84-85822-83-8 (fascículo) 84-7598-005-8 (tomo 4)
84-85822-82-X (obra completa)
Depósito Legal: B. 52/1984

Fotocomposición: Tecfa, S.A., Pedro IV, 160, Barcelona-5
Impresión: Cayfosa, Santa Perpètua de Mogoda (Barcelona) 268409
Impreso en España - Printed in Spain - Septiembre 1984

Editorial Delta, S.A., garantiza la publicación de todos los fascículos que componen esta obra.

Distribuye para España: Marco Ibérica, Distribución de Ediciones, S.A., Carretera de Irún, km 13,350. Variante de Fuencarral, 28034 Madrid.
Distribuye para Argentina: Viscontea Distribuidora, S.C.A., La Rioja 1134/56, Buenos Aires.
Distribuye para Colombia: Distribuidoras Unidas, Ltda., Transversal 93, n.° 52-03, Bogotá D.E.
Distribuye para México: Distribuidora Intermex, S.A., Lucio blanco, n.° 435, Col. San Juan Tlihuaca, Azcapotzalco, 02400, México D.F.
Distribuye para Venezuela: Distribuidora Continental, S.A., Edificio Bloque Dearmas, final Avda. San Martín con final Avda. La Paz, Caracas 1010.

Pida a su proveedor habitual que le reserve un ejemplar de MI COMPUTER. Comprando su fascículo todas las semanas y en el mismo quiosco o librería, Vd. conseguirá un servicio más rápido, pues nos permite realizar la distribución a los puntos de venta con la mayor precisión.

Servicio de suscripciones y atrasados (sólo para España)

Las condiciones de suscripción a la obra completa (96 fascículos más las tapas, guardas y transferibles para la confección de los 8 volúmenes) son las siguientes:

a) Un pago único anticipado de 16 690 ptas. o bien 8 pagos trimestrales anticipados y consecutivos de 2 087 ptas. (sin gastos de envío).

b) Los pagos pueden hacerse efectivos mediante ingreso en la cuenta 6.850.277 de la Caja Postal de Ahorros y remitiendo a continuación el resguardo o su fotocopia a Editorial Delta, S.A. (Paseo de Gracia, 88, 5.°, 08008 Barcelona), o también con talón bancario remitido a la misma dirección.

c) Se realizará un envío cada 12 semanas, compuesto de 12 fascículos y las tapas para encuadernarlos.

Los fascículos atrasados pueden adquirirse en el quiosco o librería habitual. También pueden recibirse por correo, con incremento del coste de envío, haciendo llegar su importe a Editorial Delta, S.A., en la forma establecida en el apartado b).

Para cualquier aclaración, telefonear al (93) 215 75 21.

No se efectúan envíos contra reembolso.





Denominador común

El BASICODE proporciona un nuevo procedimiento para solucionar el eterno problema de la portabilidad del software para micros



Ian McKinnell

Estándar común
El BASICODE permite que los micros se comuniquen entre sí a través de un estándar común. Utiliza un juego mínimo de instrucciones de BASIC y su propio formato de cinta para permitir que alrededor de una docena de micros puedan intercambiar sus programas. Los programas en BASICODE se transmiten incluso por emisoras de radio, posibilitando que los oyentes que poseen micros diferentes puedan ejecutar los mismos programas

En la actualidad el BASIC ya está firmemente establecido como el lenguaje estándar para micros personales. Sin embargo, como sabe todo usuario de ordenadores personales, existen enormes diferencias entre las versiones disponibles en el mercado. Aun en el caso de máquinas que compartan una versión común, como el BASIC Microsoft, no existe ninguna garantía de que un programa escrito en un tipo de ordenador funcione necesariamente en un modelo diferente.

Actualmente existen algunos indicios de que esta situación va a cambiar. A principios de 1984, el programa *Chip shop* de la BBC Radio empezó a transmitir programas en una versión única de BASIC denominada BASICODE; tales programas se vienen cargando y ejecutando con éxito en una amplia variedad de ordenadores personales.

El BASICODE es un nuevo enfoque para abordar el programa de la compatibilidad. Se desarrolló por primera vez en los Países Bajos para utilizarlo en *Hobbyscoop*, un programa de ciencia y tecnología producido por Teleac, una especie de Universidad a Distancia que imparte clases por televisión. Cuando *Hobbyscoop* empezó sus emisiones, en 1978, el espacio basaba sus transmisiones en las cuatro máquinas más populares en aquel entonces: el Apple, el Exidy Sorcerer, el Commodore PET y el Tandy TRS-80. Sólo podía haber una transmisión por semana para cada máquina y, como dos de estos ordenadores tenían velocidades de transmisión sumamente lentas, los oyentes tenían que soportar hasta ocho minutos de chirridos. Evidentemente este estado de cosas no era satisfactorio y a medida que iban saliendo al mercado nuevas máquinas, cada una de las cuales exigía su propia emisión, este método de transmisión de programas se volvió prácticamente inviable.

El problema fue abordado por primera vez por un aficionado entusiasta de la radio llamado Klaas Robers, que produjo la primera versión de BASICODE. Éste se basaba en un subconjunto común de instrucciones de BASIC, que resultaba comprensible a todo tipo de ordenadores. No obstante, persistían problemas de interpretación, ya que a pesar de que los distintos tipos de máquinas tenían instrucciones idénticas, cada una de ellas las ejecutaba de forma diferente y por ese motivo no se consiguió la estandarización. En consecuencia, Klaas Robers desarrolló, junto con Jochem Herrman, una versión mejorada del lenguaje, a la que denominaron BASICODE-2.

Las primeras emisiones de BASICODE-2 tuvieron lugar el día de Año Nuevo de 1983 y pronto demostró ser un éxito. Oyentes de lugares tan lejanos como Bélgica, Francia, Gran Bretaña, Alemania y Dinamarca informaron del éxito obtenido al cargar

los programas. Esta atención internacional aumentó cuando el servicio de transmisión holandés empezó a emitir BASICODE-2 por su red externa.

El BASICODE se basa en las 42 palabras clave y 11 símbolos que poseen en común la mayoría de las máquinas que utilizan el lenguaje. Una palabra clave de BASIC no se almacena como los caracteres que la componen sino que se almacena en forma de un código identificativo de un solo byte, cuyo valor representa la palabra clave. Por ejemplo, la palabra clave LEFT$ en el Commodore 64 se almacena en un único byte que contiene el valor 200, en vez de ocupar cinco bytes que contengan los valores ASCII para los caracteres L, E, F, T y $. Esto hace que el trabajo del intérprete de BASIC sea mucho más eficiente y emplee mucha menos memoria RAM. No obstante, a pesar de que todos los ordenadores utilizan este sistema de códigos identificativos para almacenar e interpretar un programa BASIC, cada máquina emplea códigos distintos para sus palabras clave. El problema se resolvió suministrando dos programas de traducción, el BASICODE-Save y el BASICODE-Load. Después de escribir un programa en BASIC, se guarda (SAVE) utilizando el programa BASICODE-Save, que sustituye los códigos estándar de BASICODE por los que utiliza el propio ordenador, almacenando en cinta un programa BASICODE estándar. De esta forma, el programa se puede cargar en otra máquina empleando el programa BASICODE-Load, que sustituye los códigos de las palabras clave de BASIC por la versión BASICODE.

Esto plantea una cuestión trascendental: cómo asegurar que los diversos tipos de ordenador lean y escriban en cinta de la misma forma. Nuevamente, a pesar de que las máquinas utilizan el mismo principio para cargar (LOAD) y guardar (SAVE) programas en cinta, en la práctica una cinta de programas producida por un ordenador puede muy bien ser diferente de la cinta producida por otra máquina. No sólo los datos se pueden escribir y leer en la cinta a distintas velocidades de transmisión, sino que también los bits de comienzo y de final (los indicadores que le dicen al ordenador dónde empiezan y terminan los datos), y los métodos de control de paridad (el sistema por el cual la máquina verifica que los datos se han transmitido correctamente), podrían asimismo diferir de forma radical. La solución adoptada fue suprimir los métodos de manipulación de cinta particulares de cada máquina e imponer un formato de código de audio, o audiocódigo, común para la transmisión.

En este formato los datos se transmiten a 1 200 bits por segundo. Cada byte de datos se precede por un bit de comienzo (valor 0) y se transmite empezando por el bit menos significativo y se termina con dos bits de final (ambos con un valor de 1). Por ejemplo, el valor ASCII de "A" es 65 (01000001 en binario) y en audiocódigo éste se transmitiría como 01000001011. Una marca especial de inicio, compuesta por una secuencia de bits de final transmitida durante cinco segundos, indica el comienzo de un programa BASICODE. A ésta le sigue el código de *principio de texto* (82 en hexadecimal). El programa BASICODE va seguido por un byte de control de paridad, que permite al ordenador verificar la exactitud de los datos transmitidos. Otra secuencia de cinco segundos de bits de final indica el fin de la transmisión de datos.

Aunque casi todas las máquinas se pueden adaptar a BASICODE por software, los modelos I y III del TRS-80 y el Video Genie requieren el uso de una pequeña interface con el fin de que las cintas se puedan leer correctamente. El manual suministrado con la cassette de BASICODE-2 da toda clase de detalles acerca de cómo construir la interface. A quien no desee construírsela él mismo, el grupo de usuarios de TRS-80 de los Países Bajos le puede proporcionar la placa de circuito impreso necesaria.

Para poder escribir un programa en BASICODE, primero se debe cargar el programa BASICODE-Save. Este programa no sólo permite guardar el código recién escrito en cassette de forma estándar, sino que proporciona además una lista de subrutinas exclusivas para esa máquina en particular. Estas rutinas están entre las líneas 0 y 999 y, por tanto, no se hallan disponibles para el programador.

La razón por la que el programa de traducción BASICODE-2 proporciona estas rutinas es porque una instrucción común a varias máquinas (como CLS, la instrucción para limpiar la pantalla) se podría ejecutar de distintas maneras. En lugar de utilizar la instrucción CLS, el programador emplea GOSUB 100, que hace referencia a la subrutina en BASICODE que realiza esta función.

La primera línea escrita por el programador deberá ser de la forma siguiente:

```
1000 A = (valor): GOTO 20: REM nombre del programa
```

donde (valor) es el número máximo de caracteres que utilizan todas las series juntas. De ahí en adelante, el usuario tiene libertad para programar

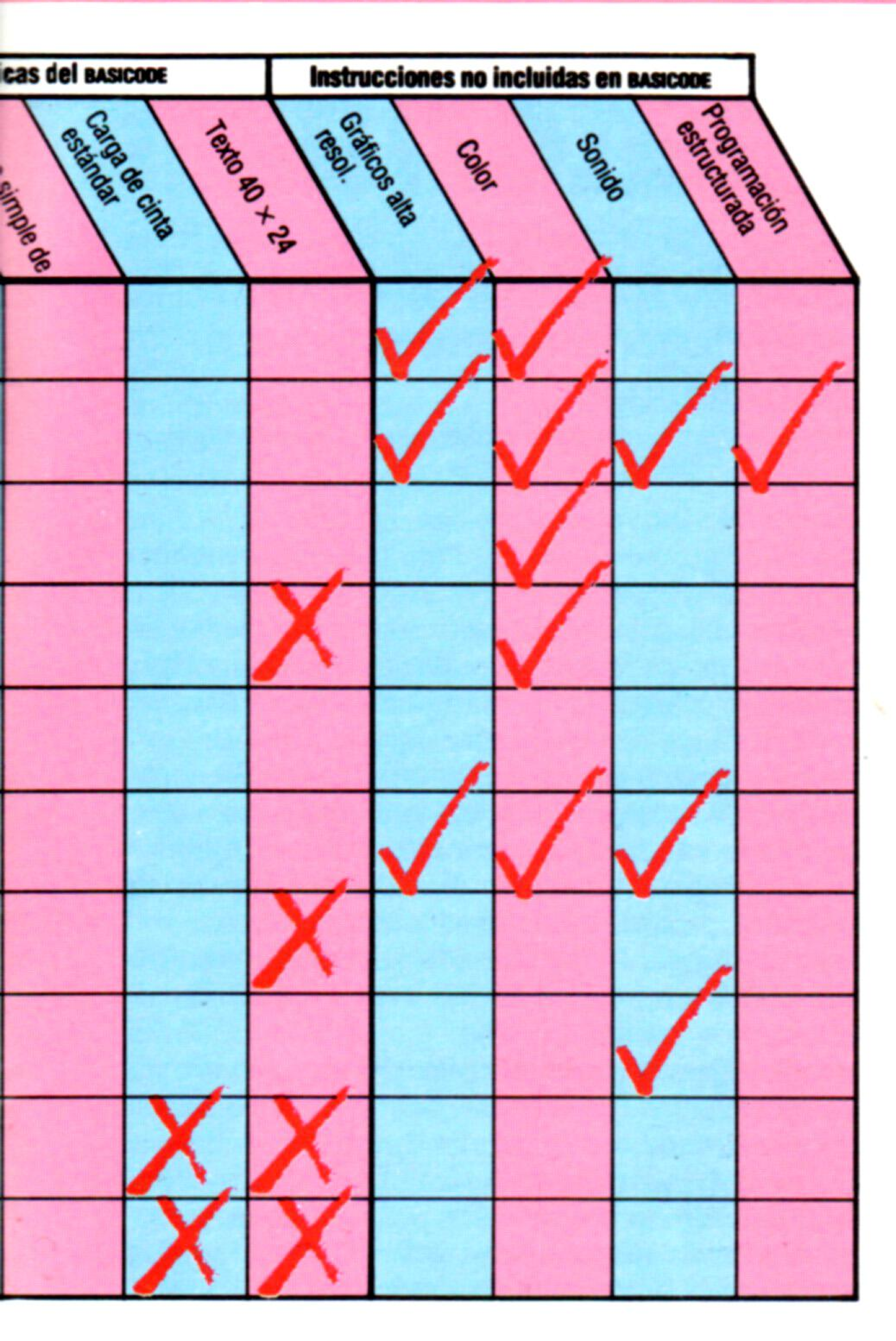
Este diagrama muestra cómo se adapta al estándar cada una de las máquinas a las que va dirigido el BASICODE. Aquellos micros cuyas especificaciones no responden al estándar están marcados con una cruz. Las máquinas marcadas con un trazo poseen características que superan lo permitido por el BASICODE



Instrucciones de operación

Ésta es una lista de las instrucciones y operaciones permitidas en BASICODE. Observe que muchas máquinas tendrán gran cantidad de palabras clave que el BASICODE no reconoce

ABS	NEXT
AND	NOT
ASC	ON
ATN	OR
CHR$	PRINT
COS	READ
DATA	REM
DIM	RESTORE
END	RETURN
EXP	RIGHT$
FOR	RUN
GOSUB	SGN
GOTO	SIN
IF	SQR
INPUT	STEP
INT	STOP
LEFT$	TAB
LEN	TAN
LET	THEN
LOG	TO
MID$	VAL
+	<
−	>
*	<>
/	<=
^	>=



como desee. Existen, sin embargo, ciertas restricciones impuestas por el formato del código. Por ejemplo, las variables se deben inicializar antes de realizar alguna operación con ellas, de modo que, por ejemplo, antes de ejecutar la orden LET T = T + 1, se debe poner T a cero.

Existen también limitaciones en cuanto al uso de varias palabras clave de BASIC. Por ejemplo:

```
5000 INPUT "CONTRASEÑA?";A$
```

está prohibido en BASICODE-2. El formato correcto de esta línea es:

```
5000 PRINT "CONTRASEÑA?": INPUT A$
```

Además, la longitud de una línea de programa no puede exceder los 60 caracteres y se asume que el tamaño de la pantalla es de 24 líneas de 40 caracteres.

Llegados a este punto vale la pena considerar el porqué de estas restricciones. A fin de incluir dentro del espectro del BASICODE la mayor cantidad posible de máquinas, el diseño se enfocó desde el punto de vista del "denominador común más bajo". Inevitablemente, hubo una compensación entre la sofisticación del BASICODE y el número de ordenadores capaces de hacer uso del mismo. Esto determinó un cierto "recorte" de sus posibilidades para acercarse a las máquinas menos potentes, lo cual se convirtió en limitaciones a las que hubo que someter a los modelos más avanzados.

Por ejemplo, un cierto número de características que el usuario de un micro personal tendría en cuenta a la hora de comprar un ordenador no están contempladas en el formato del BASICODE. El sistema no tiene incorporada ninguna facilidad para variar el tono y la duración de los sonidos. En este sentido sólo se puede trabajar con la instrucción BEEP, que es más bien primitiva. De modo similar, el BASICODE sólo permite que el programador realice gráficos en la modalidad de baja resolución. Y aun entonces éstos sólo se pueden programar en blanco y negro.

Otro problema es que desde que se inventó el BASICODE se ha producido un enorme adelanto en el desarrollo de las técnicas de programación estructurada del BASIC. El BASICODE no admite sentencias tales como WHILE... WEND, ni siquiera una instrucción DEF FN. La estructuración del programa se deja en manos de la instrucción GOSUB, de la que depende en gran medida el diseño de éste.

Por otra parte, es importante destacar que a pesar de estas restricciones, algunas de las máquinas soportadas por el BASICODE son incapaces de cumplir siquiera los estándares establecidos más modestos. Por ejemplo, el Vic-20, el ZX81, el TRS-80 y el Video Genie tienen visualizaciones inferiores a la estándar de 40×24 caracteres.

No obstante, para el programador aplicado la programación en BASICODE debería ser un desafío. Debido a que las reglas son restrictivas, el programador ha de tener un gran cuidado, asegurándose de que el programa escrito sea portable. El programador debe ceñirse a las aproximadamente 50 palabras clave y operadores que utiliza el lenguaje y usar instrucciones GOSUB para reemplazar las instrucciones no estandarizadas, como CLS. También debe tener siempre presente que algunas de las máquinas a las que se destina el BASICODE tienen una capacidad de memoria muy limitada, y los programas largos, aunque sean totalmente válidos en BASICODE, no cabrían en la RAM disponible de algunas máquinas. Sería muy posible escribir un programa que funcionase en su propio ordenador, pero la prueba de fuego sería guardarlo (SAVE) e intentar ejecutarlo sobre un ordenador distinto.

Dentro del programa principal el programador podría desear añadir ciertas mejoras no permitidas. Lo que se aconseja hacer es usar sentencias REM que expliquen con exactitud qué es lo que el programador se propone realizar. Los autores del BASICODE recomiendan escribir estas sentencias entre las líneas 20000 y 24999, si bien no es obligatorio. Después de cargar el programa, el usuario puede realizar las mejoras descritas adecuadas a su propia máquina.

Al paquete que proporciona la BBC para utilizar con la serie de transmisiones *Chip shop* se adjuntan instrucciones acerca del uso del BASICODE-2. El usuario recibe una cassette con los programas de traducción para las diversas máquinas en la cara uno. Aunque la mayoría de los ordenadores exigen la carga de un único programa de traducción a BASICODE-2, los micros BBC y el Vic-20 poseen programas SAVE y LOAD separados. Todos los programas incluyen instrucciones habladas para ayudar al usuario a encontrar el programa adecuado. En la cara dos hay 18 programas de demostración para dar idea de las posibilidades del BASICODE.

Dadas las inmensas variaciones en cuanto a lo que se supone que es un lenguaje estándar, los autores del BASICODE han conseguido un notable éxito al reunir tantas máquinas bajo el mismo alero.



Ian McKinnell

El manual y la cassette de BASICODE-2 se pueden adquirir enviando un cheque o giro postal por valor de 3,95 libras a nombre de Broadcasting Support Services a:

Broadcasting Support Services
P.O. Box 7
London
W3 6XJ
Gran Bretaña

Selección aleatoria

Los archivos aleatorios ofrecen un acceso mucho más rápido a cambio de consumir más espacio de almacenamiento y requerir un diseño más elaborado que los secuenciales

Las limitaciones de los archivos secuenciales surgen debido a la necesidad de leer la información almacenada en ellos por el orden en que se grabó. Los dispositivos de acceso aleatorio o directo dan solución a estas limitaciones porque los registros que contienen pueden ser accedidos en cualquier orden y muy rápidamente. La palabra «aleatorio» significa que en el archivo se puede escribir o leer cualquier registro de datos sin necesidad de pasar a través de toda la información precedente.

El problema obvio es que todos los archivos contenidos en cinta de cassette deben ser archivos secuenciales. No existe forma alguna de ir directamente a un dato que esté en la mitad de la cinta de cassette, siendo inevitable la lectura de toda la cinta hasta ese punto. La única forma de utilizar archivos de acceso directo en un micro basado en el almacenamiento en cinta consiste en cargar todos los datos en la memoria, aunque esto limite el tamaño del archivo. Para un acceso aleatorio verdaderamente útil son necesarias las unidades de disco, y aun así, unas pocas marcas de unidades de disco no admiten este tipo de tratamiento de archivos.

El usuario notará también que es más sencillo trabajar con archivos de acceso aleatorio que usar las técnicas más bien incómodas que se precisan para los archivos secuenciales. La división del archivo en registros y campos que detallamos en la página 706 es muy importante en el caso de los archivos de acceso aleatorio. Para acceder al archivo es necesario especificar el registro requerido. Este registro será colocado entonces en un buffer de la memoria del ordenador, donde se pueden eliminar, actualizar o imprimir los campos que contiene.

Afortunadamente, el sistema operativo se ocupará de gestionar las estructuras necesariamente más complicadas que requiere este tipo de ficheros. Con objeto de facilitar su rápida localización, todos los registros de un archivo directo tienen la misma longitud. Si cada uno tuviera, por ejemplo, 100 bytes o caracteres de longitud, y el programa solicitara el registro número 83, el sistema operativo posicionaría el cabezal del disco sobre el byte 8300 del archivo. Puesto que el sistema operativo sabe cuántos bytes hay en cada sector del disco, no es difícil calcular en qué sector del disco se halla el registro requerido. Este método de lectura de archivos podría parecer complicado, y por cierto es lento; pero, a pesar de todo, es mucho más rápido que leer a través de un archivo secuencial.

Al normalizar la longitud de los registros, obviamente es necesario elegir un tamaño que sea suficiente para contener el registro más largo almacenado en el archivo. Los registros más cortos se rellenan, por lo general, con espacios (32, en código ASCII). Éste es un problema importante de los archivos aleatorios, ya que el relleno requerido para dejar todos los registros con la longitud elegida implica desperdiciar un precioso espacio de almacenamiento. Esto significa que los archivos aleatorios se utilizan para pequeñas cantidades de información pero cuyo acceso deba ser muy rápido, mientras que los archivos secuenciales se utilizan para el almacenamiento masivo de datos en los que la velocidad de acceso no sea demasiado importante.

Los campos de un registro también se deben definir con una longitud fija. Esto es particularmente relevante para los sistemas que proporcionan facilidad de acceso directo a campos además de registros. Aun para los sistemas que no disponen de esta facilidad, sigue siendo una forma elegante y eficaz de definir sus archivos. El primer paso al diseñar la estructura de los registros de un archivo de acceso aleatorio consiste en estudiar los distintos campos y decidir las longitudes idóneas para los mismos.

La economía es crucial al diseñar un archivo, ya que necesariamente habrá una compensación entre la cantidad de información almacenada y el número de registros distintos. Con bastante frecuencia pueden elaborarse sistemas de codificación que reduzcan la cantidad de espacio ocupado por los datos. Por ejemplo, códigos de color 1, 2 y 3 para representar los colores negro, rojo y verde, o códigos como 841011 para representar la fecha 11 de octubre de 1984. Los sistemas de codificación deben ser

Aleatorio contra secuencial

	ARCHIVOS DE ACCESO DIRECTO	ARCHIVOS SECUENCIALES
PROS	• Rápido acceso a registros determinados	• Conservan espacio • Disponibles para sistemas basados en cintas
CONTRAS	• Desaprovechan espacio • Necesitan discos	• Lentos y engorrosos
APLICACIONES ADECUADAS	• Datos predecibles que estén en un formato definido • Cuando se accede a un pequeño número de registros diferentes; por ejemplo, en una biblioteca donde los clientes solicitan detalles de libros determinados	• Grandes cantidades de datos sin estructurar • Cuando la mayoría de los registros de un archivo se procesan en una única ejecución del programa; por ejemplo, en un sistema de salarios, donde se debe pagar a cada empleado

internos del sistema, y los programas deben convertir los códigos a una forma fácilmente comprensible al visualizarlos por pantalla el operador.

Existen otras dos consideraciones que hay que tener presente al determinar las longitudes de los registros. La mayoría de los sistemas limitan la longitud máxima disponible. Ésta puede variar normalmente desde 128 hasta 2 048 bytes. Además, suele ser más eficaz elegir una longitud que sea un múltiplo del tamaño del sector (cifras como 64, 128, 256 o 512 son las más comunes). Esto evita que los registros individuales queden a caballo de dos o más sectores y, por consiguiente, reduce el número de accesos a disco.

Los archivos directos, por lo general, son más fáciles de tratar que los archivos secuenciales. En ambos sistemas es necesario mantener un contador actualizado del número de registros del archivo, y muy a menudo se usa en archivos directos el registro 0 para guardar esta información y otra información relevante, como la fecha de creación del archivo. Obviamente, este registro tiene una estructura de campos distinta a la del resto.

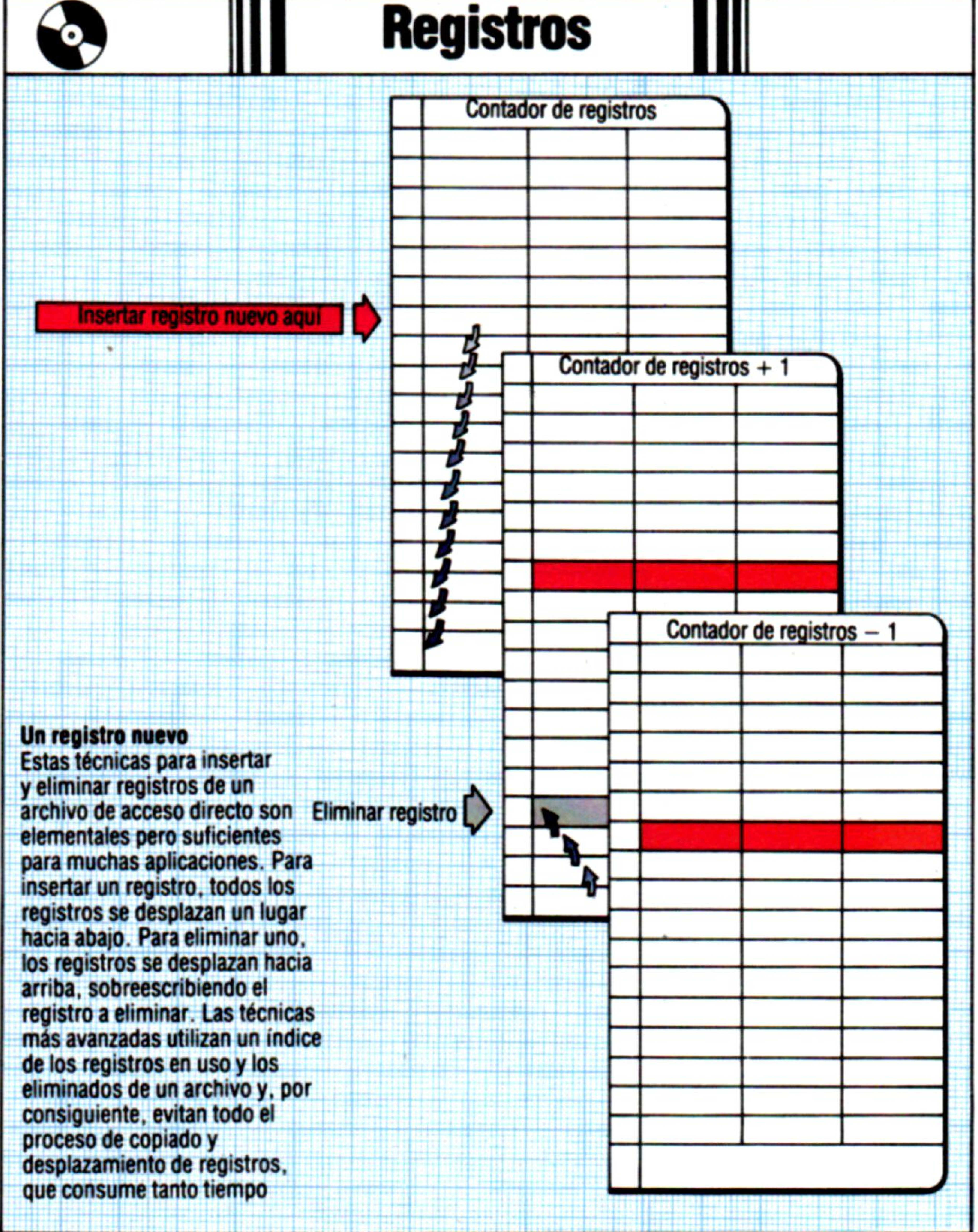
Un registro se puede actualizar leyéndolo, modificándolo y volviéndolo a escribir luego en su sitio. El registro es accedido por su número. Obviamente, no es razonable pedirle al usuario que recuerde el número correspondiente a cada registro. De modo que existe una amplia variedad de técnicas para buscar y localizar registros determinados, similares conceptualmente a las técnicas que se utilizan para acceder a elementos contenidos en matrices de BASIC. Con frecuencia se usa un campo determinado, por ejemplo, un campo de nombre, como clave de acceso del archivo. El ordenador lee el campo de clave y elabora un índice que identifica dónde están almacenados varios nombres.

En los archivos directos sin índice se suele buscar registro a registro, al igual que en los archivos secuenciales. Sin embargo, si los registros están clasificados por el campo clave, se pueden utilizar métodos de búsqueda rápidos. Supongamos, por ejemplo, que deseamos buscar "Jiménez" en un archivo clasificado por apellido. Comenzamos buscando el registro que está en la mitad del fichero y descubrimos que el apellido es "Paredes". Alfabéticamente, "Jiménez" está antes que "Paredes", de modo que podemos descartar todo lo que haya después de este registro. Nuestro siguiente intento es, entonces, un registro que esté a medio camino de la primera mitad del archivo. El siguiente apellido podría ser "Hernández", en cuyo caso deberemos volver a repetir la operación, y así sucesivamente. Esta clase de técnica puede ser muy compleja, y muchos programas mejoran el rendimiento teniendo en RAM un gran número de los registros utilizados más frecuentemente, de modo que estén disponibles con rapidez. Como resultado, en archivos grandes se localizan y almacenan registros a gran velocidad.

Eliminar e insertar registros nuevos puede ser comparativamente lento. El método más burdo para eliminar un registro consiste en copiar, en el espacio que ocupa, el registro que viene inmediatamente tras él, sobreescribiendo por consiguiente la información que contenía. Todos los registros subsiguientes se copian entonces una posición hacia arriba y, por último, el contador de registros se reduce en uno. De forma similar, se puede insertar un registro nuevo en cualquier punto desplazando un lugar todos los registros desde ese punto hasta el final del archivo. Esto crea un "agujero" de un registro en donde se puede colocar el registro nuevo.

Ninguna de estas técnicas es rápida, si bien son más eficaces que las operaciones similares con archivos secuenciales. No obstante, las inserciones y eliminaciones se pueden efectuar de manera mucho más expedita si el archivo posee un índice separado. Cuando se elimina un registro, se lo marca como eliminado en el índice, sin necesidad de modificar sus datos. A medida que se añaden nuevos registros, éstos pueden ser ubicados en los espacios correspondientes a registros eliminados o sin usar y actualizar el índice.

Existen dos ventajas finales a considerar en el sistema de archivos aleatorios o directos. En primer lugar, mientras que ciertamente es más rápido leer y escribir grupos de registros juntos, los archivos pueden quedar desordenados. Por lo tanto, la mayoría de los programas ofrecen una facilidad de reorganización que clasifica los registros por un orden lógico y descarta los registros eliminados. En segundo lugar, el sistema de marcar meramente los registros eliminados como borrados proporciona un útil mecanismo de seguridad, ya que es fácil recuperar estos registros en caso de ser necesario. Este mecanismo de seguridad será válido hasta el momento en que se ejecute un programa de reorganización sobre este fichero.

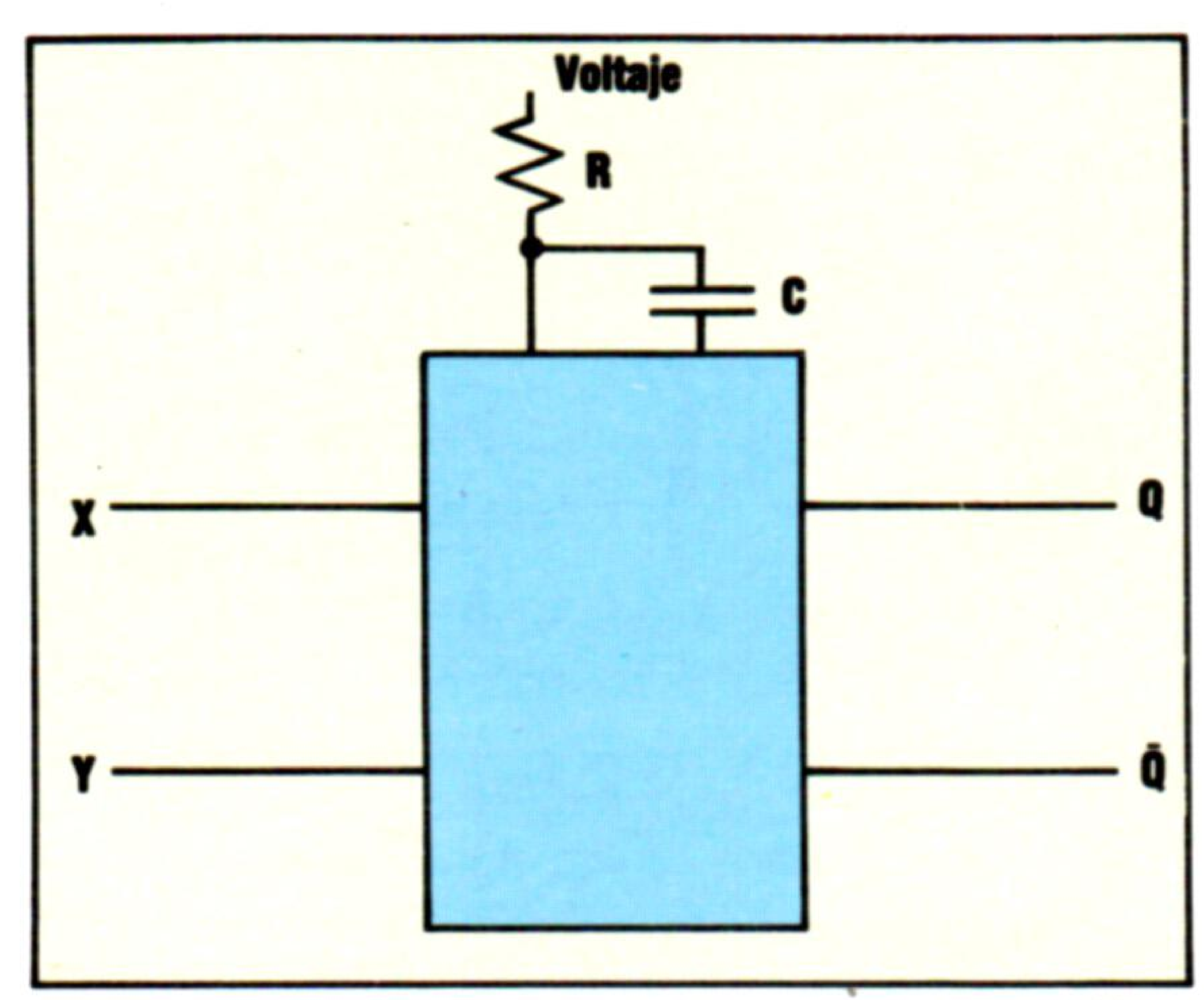


Un registro nuevo
Estas técnicas para insertar y eliminar registros de un archivo de acceso directo son elementales pero suficientes para muchas aplicaciones. Para insertar un registro, todos los registros se desplazan un lugar hacia abajo. Para eliminar uno, los registros se desplazan hacia arriba, sobreescribiendo el registro a eliminar. Las técnicas más avanzadas utilizan un índice de los registros en uso y los eliminados de un archivo y, por consiguiente, evitan todo el proceso de copiado y desplazamiento de registros, que consume tanto tiempo

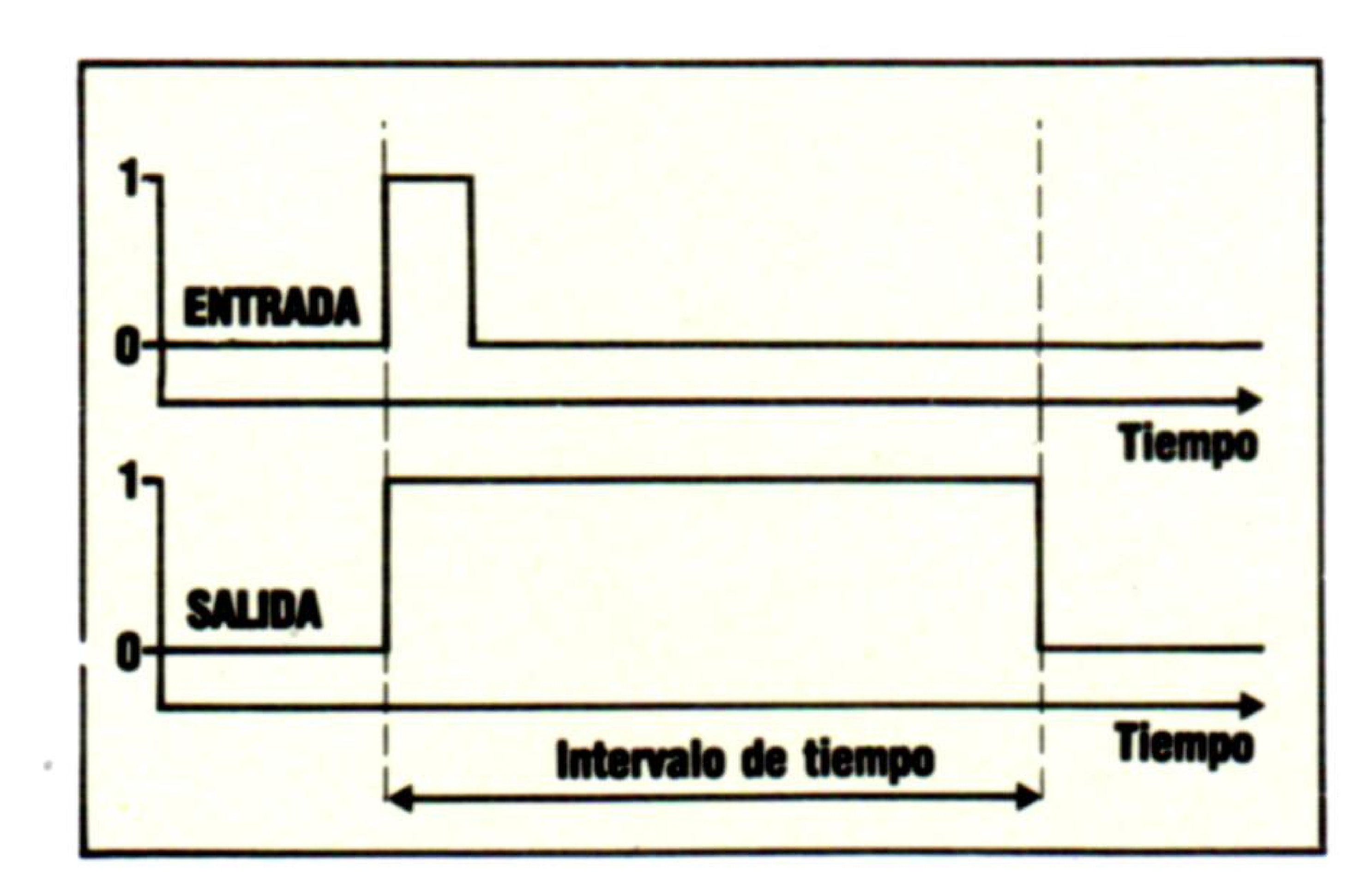
Marcando el compás

Para controlar sus funciones, un ordenador requiere una sincronización exacta. Analicemos tres circuitos que colaboran en esta tarea: el monoestable, el flip-flop tipo D y el flip-flop J-K

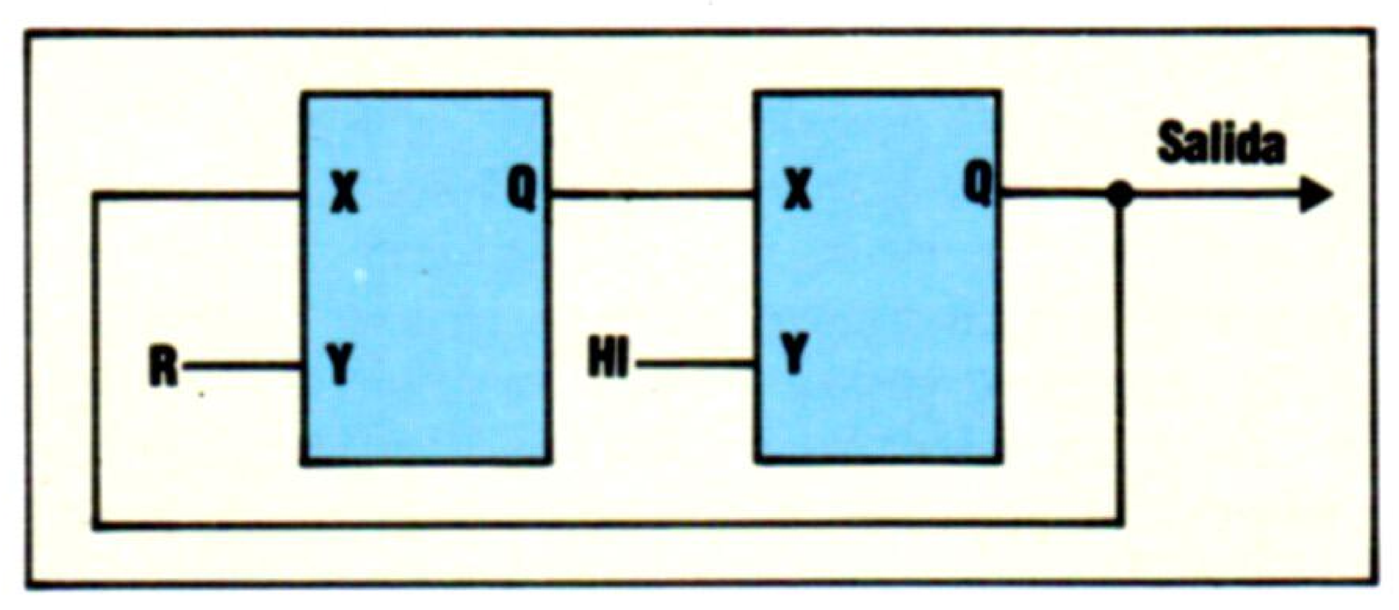
Un *circuito monoestable* proporciona un medio para introducir intervalos fijos de tiempo en las operaciones de los circuitos lógicos. Cuando un circuito monoestable recibe un impulso de entrada, la salida se pone en 1 (HI) durante un intervalo de tiempo fijo antes de volver a su estado normal (LO) de salida cero. El período de tiempo durante el cual la salida es HI está determinado por los valores de ciertos componentes del circuito. Éste es un ejemplo de circuito monoestable:



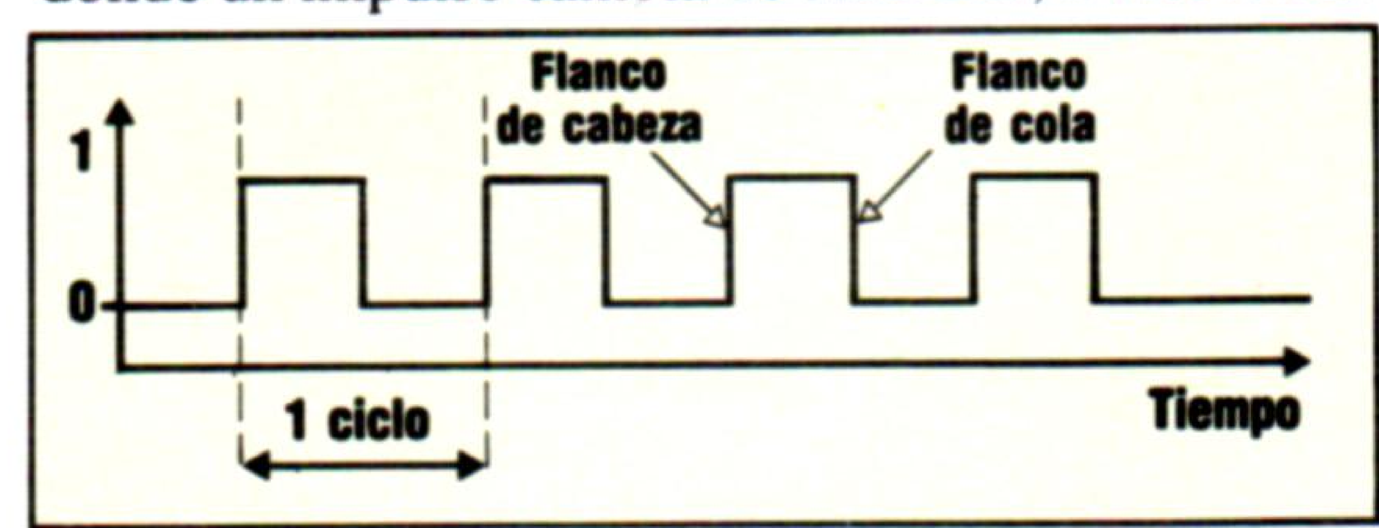
Este dispositivo se puede activar cambiando X de HI a LO o Y de LO a HI. Alterando los valores de la resistencia, R, y del condensador, C, se puede modificar el tiempo de salida. Este gráfico ilustra cómo están relacionadas la entrada y la salida:



La duración de la salida HI se podría utilizar para controlar el motor paso a paso de un lector de cinta o para retardar la transmisión de un bit durante un cierto período de tiempo. Dos circuitos monoestables se pueden unir entre sí para proporcionar un impulso de reloj, que oscila a intervalos fijos entre HI y LO:



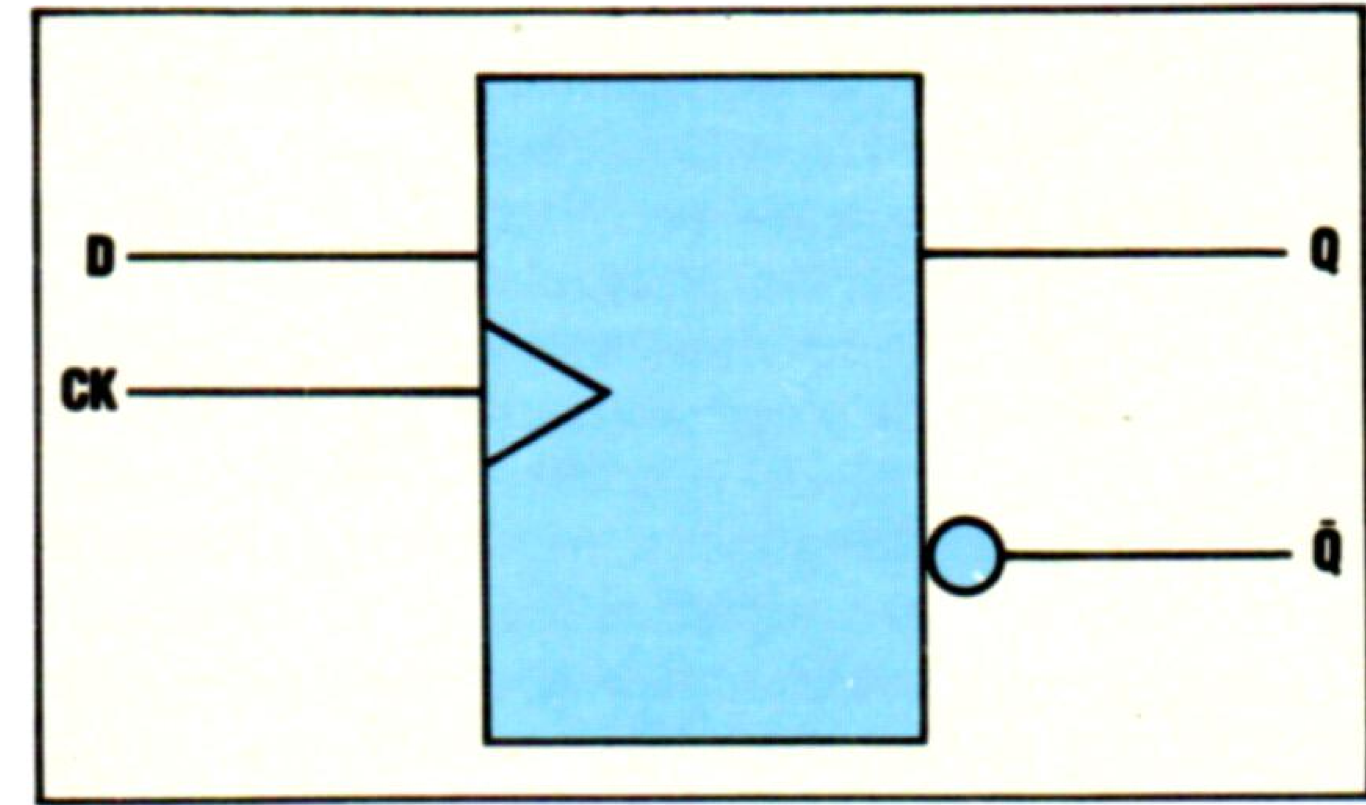
La salida producida tiene el aspecto característico de una "onda cuadrada" (como se ve en nuestros gráficos). Al intervalo de tiempo que media entre los dos inicios de la señal de salida HI se le denomina *ciclo*. Típicamente, éste es una millonésima de segundo. Esta señal de reloj continua es el latido del corazón del ordenador, y temporiza las muchas funciones que se llevan a cabo en la CPU. El siguiente diagrama indica los nombres con que se designan los "flancos" del gráfico de onda cuadrada, donde un impulso cambia de HI a LO, o viceversa:



Analicemos ahora dos nuevos tipos de flip-flop cuyas acciones están gobernadas por los impulsos regulares del reloj.

El flip-flop tipo D

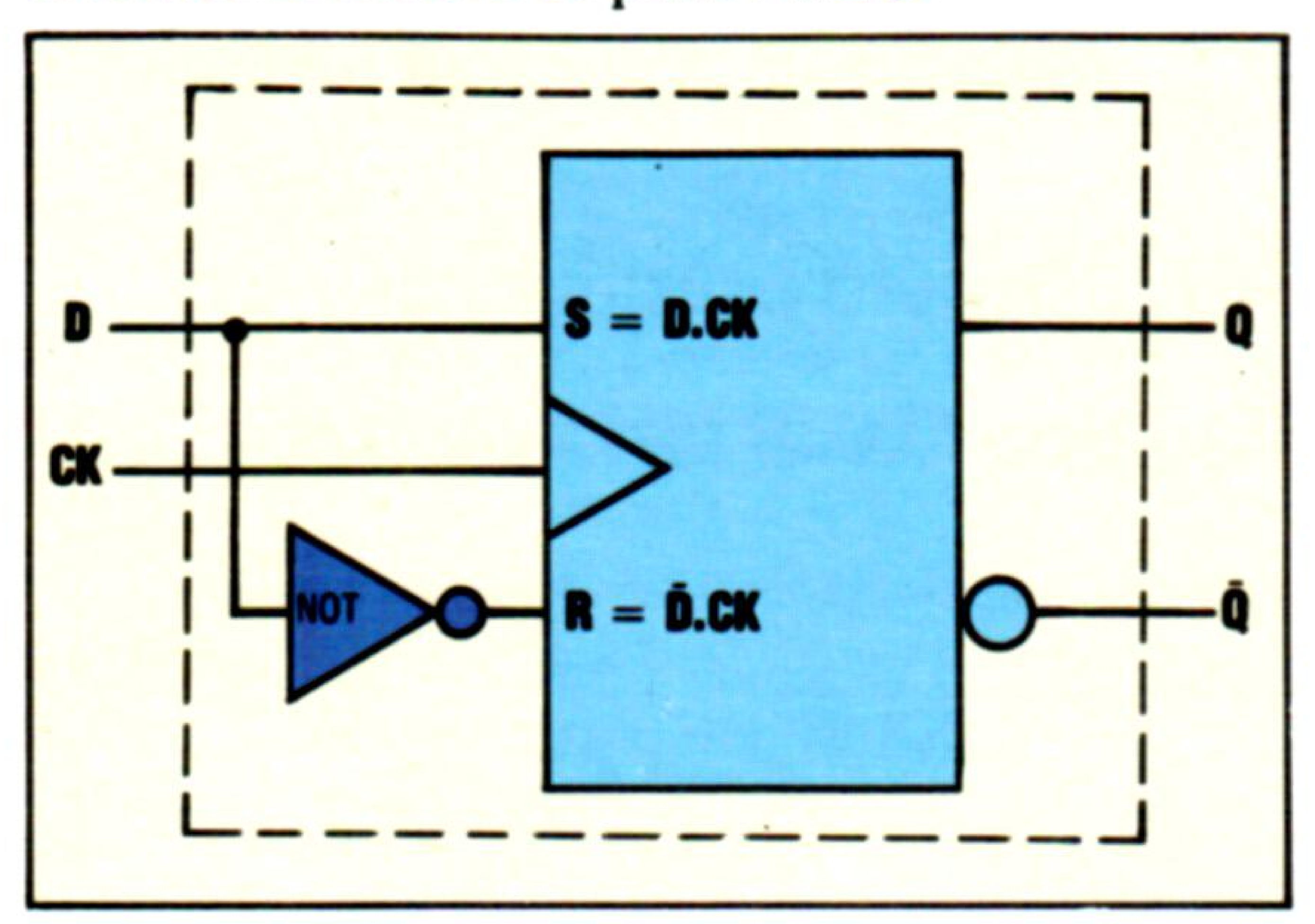
El flip-flop tipo D tiene una entrada lógica (D) y una entrada de reloj (CK):

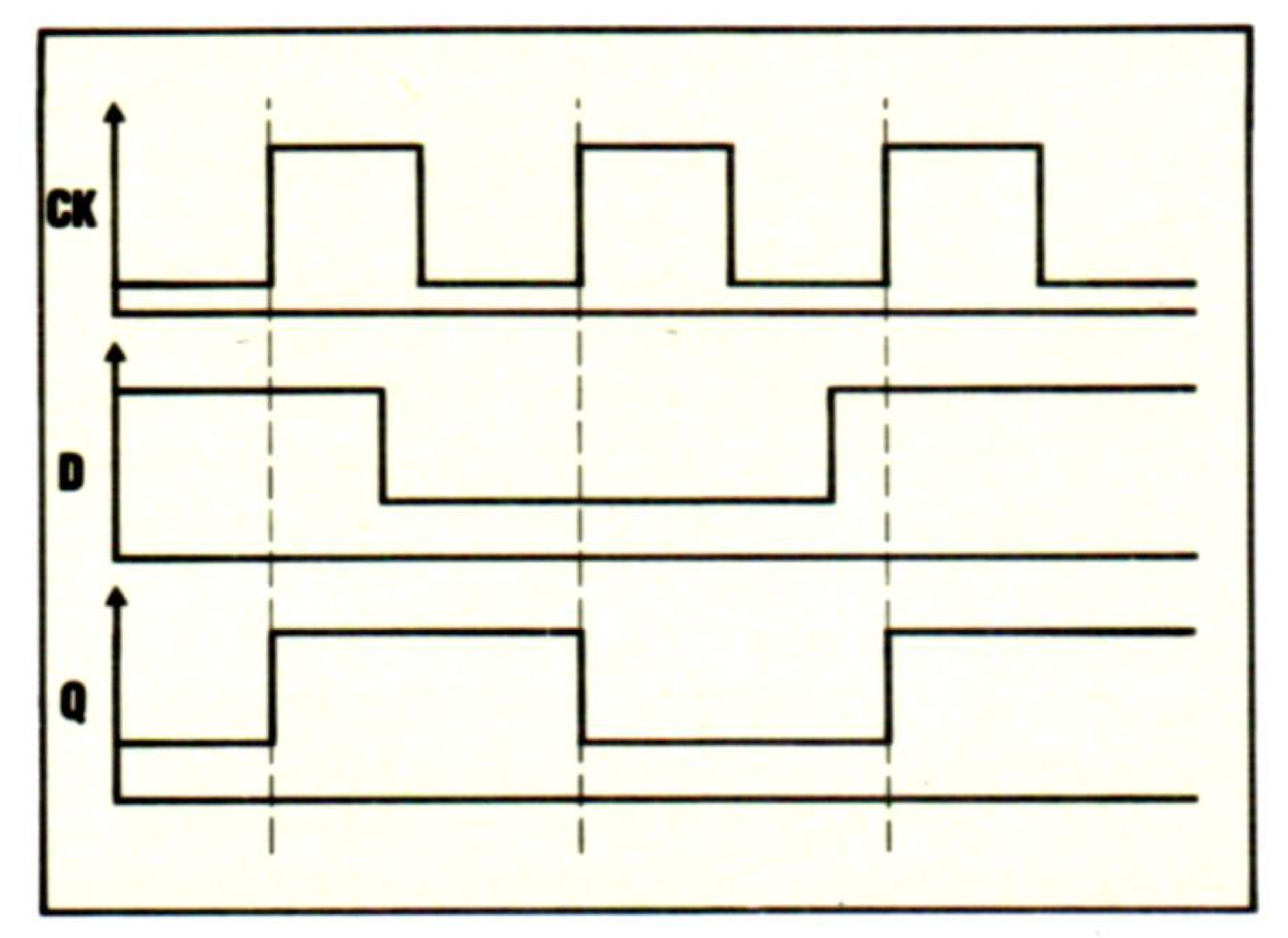


El diseño del tipo D se basa en el flip-flop R-S, que analizamos en el capítulo anterior. Sin embargo, es la entrada de la señal del reloj lo que produce el método de operación especial conocido como *latching*. La salida del circuito, Q, se determina al comienzo de un ciclo de reloj. Si en ese momento la entrada en D es HI, entonces la salida Q se pone en



HI. Si, por el contrario, la entrada en D es LO, entonces la salida Q se pone en LO.

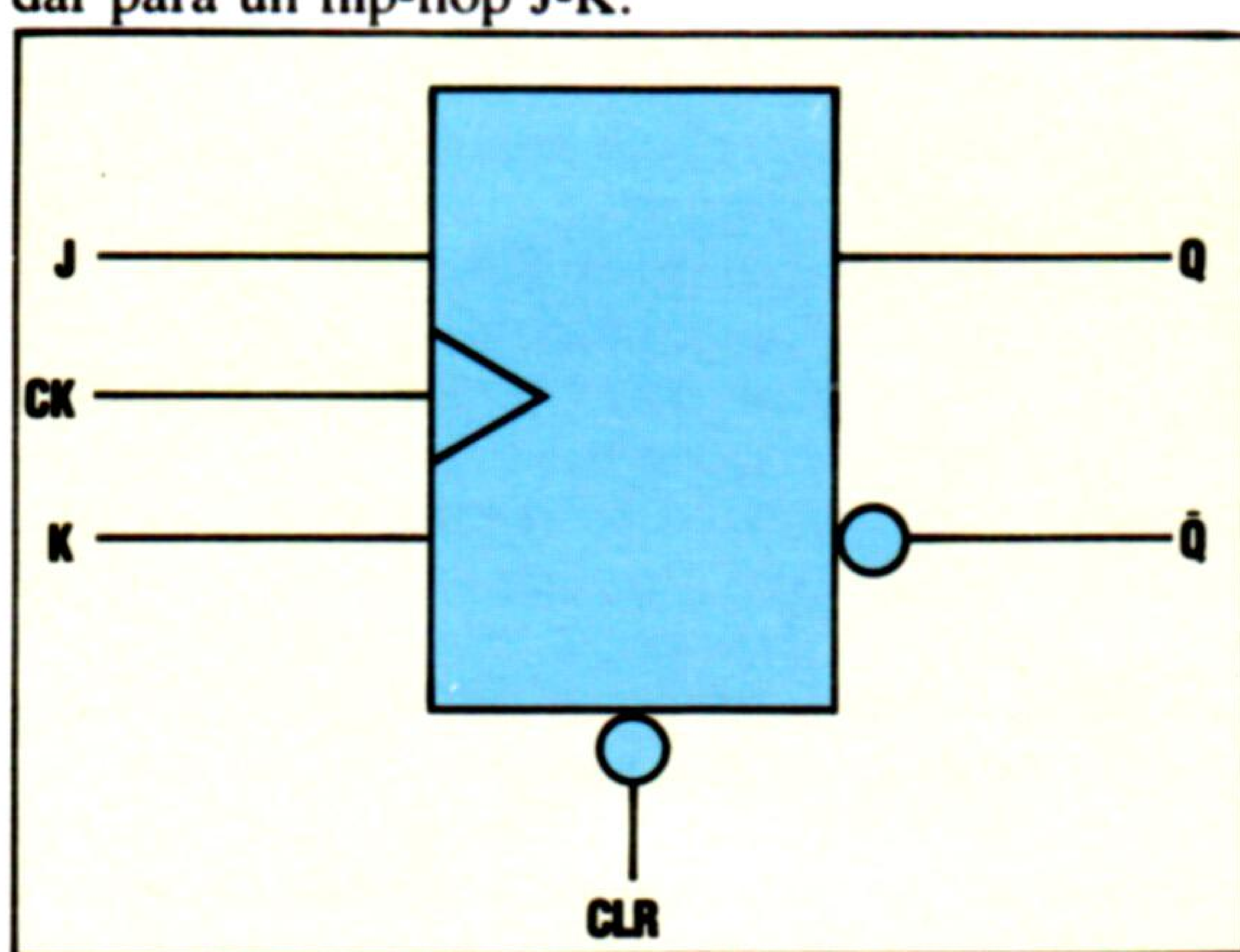




A partir de estos gráficos se puede ver que la salida Q sólo puede cambiar durante una transición del reloj de LO a HI. Por ello, el flip-flop tipo D se dice que es "activado por flanco de cabeza".

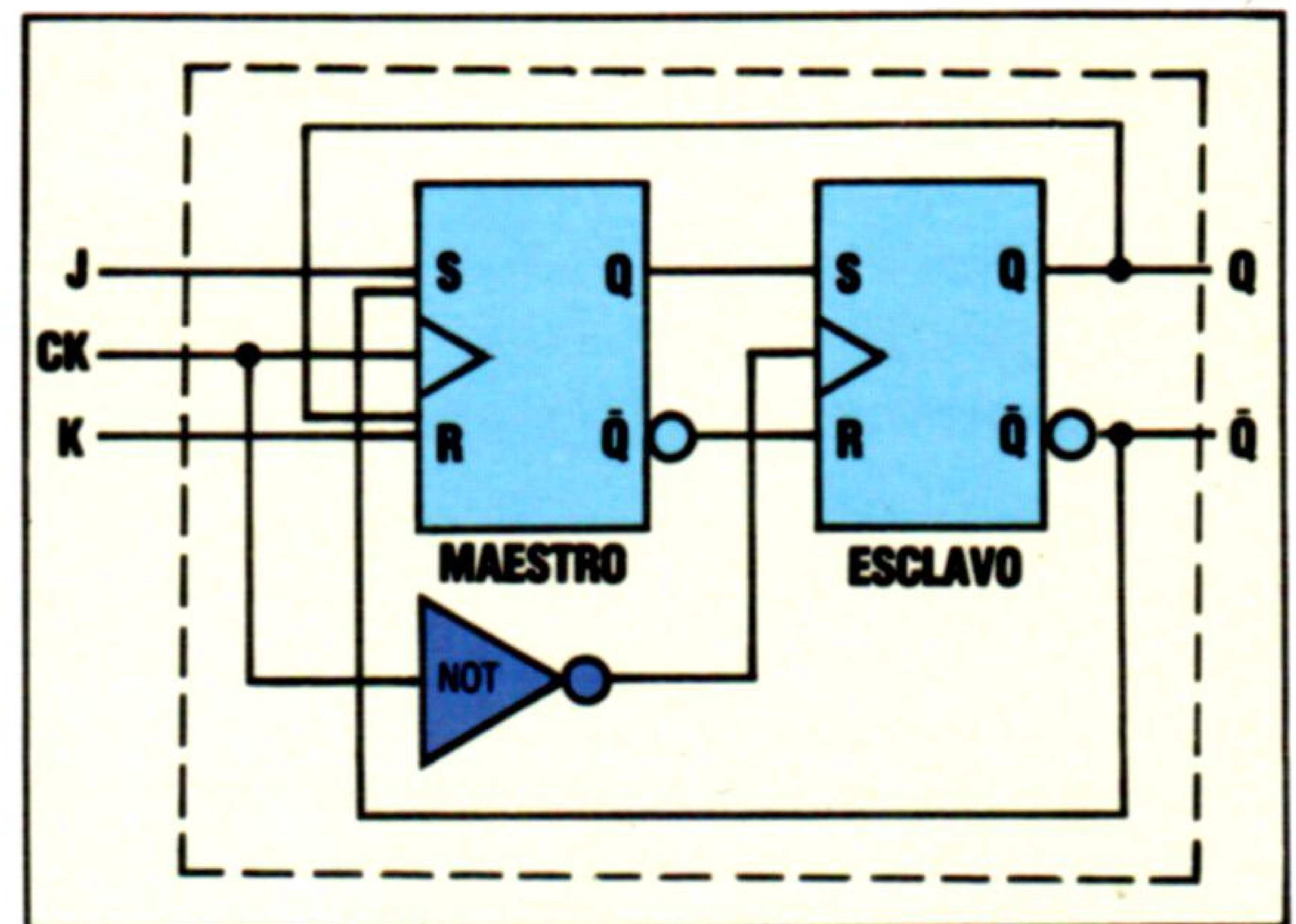
El flip-flop J-K

Se dice que el flip-flop J-K es un dispositivo maestro-esclavo porque se compone de dos flip-flops R-S en relación "dominante-sometido". Un dispositivo maestro-esclavo permite almacenar un impulso de entrada en un flip-flop, dando por la otra unidad una salida dependiente de la entrada previa, dentro de un ciclo de reloj. Un ejemplo es la operación de desplazamiento, común en la mayoría de los procesadores, en la que los bits de un registro se desplazan un lugar a la izquierda o a la derecha. He aquí el diagrama de circuito estándar para un flip-flop J-K:



Liz Dixon

El siguiente diagrama muestra cómo están enlazados entre sí los dos tipos R-S. Uno es el "maestro" y el otro el "esclavo". Supongamos que se aplica una entrada en J o K: si la señal de entrada del reloj es HI se activa el maestro; si es LO, alimenta las entradas del esclavo, pues los tipos R-S se activan por flanco de cabeza. Así, sólo un R-S está activado en cualquier momento y queda "bloqueada" la entrada previa en el otro R-S:



En el margen tenemos una *tabla de estados* para el flip-flop J-K. Ésta es similar a una tabla de verdad, pero hace uso de una variable, Q_0, la salida previa. Observe que HI entra simultáneamente por J y K, y esto hace que el flip-flop cambie de estado a cada impulso de reloj. Esto se conoce como temporizado (*toggling*) y se produce por la realimentación de las salidas esclavas a las entradas maestras. Con un flip-flop R-S ésta no es una combinación de entradas permitida, quedando indefinida la salida. Considerando Q_0, J y K como entradas, los resultados de la tabla de estados se pueden colocar en un diagrama K.

Q_0	J	K	Q
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

	Q_0	$\bar{Q}_0$	
J		1	K
$\bar{J}$			K
$\bar{J}$	1		$\bar{K}$
J	1	1	$\bar{K}$

A partir del diagrama K podemos obtener la expresión lógica:

$$Q = \bar{Q}_0.J + Q_0.\bar{K}$$

Se dice que esta ecuación es la *característica* del flip-flop J-K.

Respuestas al ejercicio 7 de la página 709

1) Al flip-flop también se lo conoce por biestable porque sólo puede estar en uno de dos estados (a saber, cuando Q=1 y Q̄=0 o cuando Q=0 y Q̄=1).

2) a) Éste no es un estado posible.

b) El flip-flop cambiará al estado RESET si la entrada superior es considerada primero, o al estado SET si la puerta inferior es considerada primero.

c) Sí (véase respuesta b).

d) Para que todos los registros adquieran un estado predecible al conectar el equipo, se les debe situar a todos en estado SET o RESET en el proceso de inicialización del sistema.

Cuentas claras

Veremos a continuación, mediante un ejemplo práctico, cómo funcionan los contadores

INICIO
COGER UNA BOLSA
SUMAR 1 AL CONTADOR DE BOLSAS
COGER UNA CANICA
ES NEGRA ? — SÍ: SUMAR 1 CONTADOR NEGRAS — NO: SUMAR 1 CONTADOR BLANCAS
HAY MÁS CANICAS ? — SÍ — NO
MÁS NEGRAS QUE BLANCAS — SÍ: N.° BOLSA NEGRAS BLANCAS — NO: N.° BOLSA BLANCAS NEGRAS
MÁS BOLSAS — SÍ: CONTADOR NEGRAS A CERO, CONTADOR BLANCAS A CERO — NO
FIN

Analicemos el siguiente problema: un niño tiene una serie de bolsas que contienen canicas de dos colores: blancas y negras. Quiere llevar un control de cuántas bolsas posee, así como del número de canicas de cada color que hay en cada bolsa. Todo ello lo escribe en un papel, indicando el número de la bolsa que ha controlado y el número de canicas en orden de color predominante. Veamos un ejemplo:

BOLSA 5: BLANCAS 10; NEGRAS 7
BOLSA 9: NEGRAS 20; BLANCAS 12

En este caso se observa que el programa va a emplear tres contadores: el de bolsas, que va incrementándose por cada nueva bolsa, y uno para cada color de canica; estos dos irán poniéndose a cero cada vez que se empiece con una nueva bolsa, ya que, de no hacerse así, al cogerse otra bolsa sin borrar el contenido previo del contador, éste indicaría el número de canicas acumulado de todas las bolsas, con lo cual se obtendría un falso resultado.

Al no saberse de antemano el número de bolsas disponibles, ni conocer la cantidad de canicas por bolsa, es preciso obtener esa información a través de consultas al operador.

En caso contrario, es decir, si no se supiera previamente cuál es el número de bolsas con que se cuenta, no sería preciso consultar al operador, por cuanto por programación se controlaría el número final de bolsas.

```
10 REM BOLSAS CON CANICAS
15 PRINT "COGER UNA BOLSA"
20 X = X + 1
25 PRINT "COGER UNA CANICA"
30 INPUT "ES BLANCA" (S/N)";A$
40 IF A$="S" THEN B=B+1 : GOTO 60
50 IF A$ <> "N" THEN GOTO 30
55 N = N + 1
60 INPUT "HAY MAS CANICAS? (S/N)";B$
70 IF B$="S" THEN GOTO 25
80 IF B$<>"N" THEN GOTO 60
90 REM DECISION COLOR PREDOMINANTE
100 IF B>N THEN PRINT "BOLSA"X, "BLANCAS"B,
    "NEGRAS"N: GOTO 120
110 PRINT "BOLSA"X, "NEGRAS"N, "BLANCAS"B
120 INPUT "HAY MAS BOLSAS? (S/N)";C$
130 IF C$="N" THEN END
140 IF C$<>"S" THEN GOTO 120
150 B=0 : N=0 : GOTO 15
```



Atractivo y ligero

El Apricot es un ordenador en tres piezas lo suficientemente ligero como para ser transportado con facilidad

Diseñado en gran parte teniendo en mente al usuario de gestión, el Apricot tiene dos versiones: una con dos unidades de disco flexible de 3 ½" y la otra con una unidad de disco flexible de 3½" y un disco fijo de 10 Mbytes. Aunque dispone de numerosas facilidades diseñadas para atraer al usuario serio, el Apricot hace pocas concesiones al mercado del ordenador personal: no posee gráficos en color, conexión para cassette, accesorios para juegos o salida para televisor. No obstante, se suministran como estándar un monitor monocromático de alta resolución, una puerta en paralelo para impresora, una puerta serial RS232, un conector para un ratón opcional, algo de software y un teclado de calidad.

Lo más llamativo del Apricot es su teclado, versátil e innovador. Una característica novedosa es la Microscreen (micropantalla): una visualización en cristal líquido de dos líneas de 40 caracteres cada una, situadas en la parte superior derecha de las teclas principales. Al conectarse el ordenador, la línea superior de la Microscreen visualiza el día de la semana, el mes, el año y la hora. La fecha y la hora se pueden alterar utilizando uno de los programas de utilidad, y un reloj a pilas mantiene la hora cuando el ordenador no se está utilizando.

Cuando se enciende la máquina comienza automáticamente un programa de autocomprobación. Éste visualiza la cantidad de memoria disponible (lo estándar son 256 Kbytes pero se pueden ampliar a 768 Kbytes) y solicita al usuario que inserte el disco maestro MS-DOS. Para los usuarios que no estén familiarizados con sistemas operativos como el CP/M o el MS-DOS, un "menú" de fácil comprensión para el usuario llamado *Manager* (director) permite la fácil selección del software de aplicaciones (como Supercalc, Multiplan, BASIC Microsoft, etc.) o utilidades (como el configurador del teclado o el editor de pantalla).

El control de la Microscreen es por software, de modo que actúa como algo más que una mera representación visual de la hora y la fecha. Existen seis teclas programables por el usuario y las funciones asignadas a las mismas se pueden mostrar en cualquier momento en la pantalla de cristal líquido. De modo, entonces, que cuando un programa visualiza un menú de opciones en la pantalla principal, el mismo menú se puede duplicar en la Microscreen. Tocar la tecla de función adecuada equivale a seleccionar el ítem del menú en pantalla pulsando las teclas Cursor y Return. La única objeción es que las teclas de membrana al estilo de las que posee el Sinclair ZX81 son menos cómodas y de uso más engorroso que las teclas convencionales tipo máquina de escribir.

Existen también ocho teclas de función ordinarias. Éstas llevan inscritas los nombres de sus funciones normales (HELP, PRINT, MENU, FINISH, etc.). No obstante, como todas las teclas del Apricot, éstas se pueden reconfigurar con el programa Keyedit suministrado. La sensación táctil del teclado del Apricot responde al elevado estándar que cabe esperar de un ordenador de gestión, pero las teclas Control y Escape están en un lugar ligeramente insólito. Para que la máquina resulte fácil de mover, el teclado se conecta a la parte inferior de la unidad principal. La pesada carga que representa el monitor separado es suficiente, no obstante, para desalentar todo intento de considerarla como realmente portátil.



Una oferta tentadora
El ACT Apricot es uno de los ordenadores de aspecto más atrayente que existen en el mercado. Al equipo también se le ha fijado un precio modesto tratándose de un micro de oficina, aunque sus especificaciones son elevadas. Utiliza un microprocesador de 16 bits con una memoria estándar de 256 K y viene con un monitor de gran calidad

Chris Stevens

El software que se proporciona con el Apricot es una amplia gama de utilidades del sistema y Supercalc. El proceso de aplicaciones numéricas tipo "¿Qué sucedería si?" se cubre con Supercalc y Superplanner.

En estos productos se detectan evidencias, al igual que en otros elementos de software adaptados para ser usados en el Apricot, de un intento bastante apresurado de tenerlos a punto justo a tiempo para el lanzamiento del ordenador. Con la máquina vienen dos sistemas operativos, MS-DOS y CP/M-86. Los usuarios del Apricot tienen prometidas copias del Concurrent CP/M-86 cuando éste se encuentre a punto.

Con el ACT Apricot sólo se suministra la versión uno del Supercalc.

Una de las primeras críticas que se le plantearon al Apricot fue que el sistema operativo MS-DOS se había implementado mal y que era lento. Ahora este problema parece haberse superado. El software de aplicaciones que se suministra con la máquina parece funcionar con razonable rapidez y los programas de comprobación (*benchmark*) para probar el MBASIC de Microsoft también son relativamente rápidos.



Microfloppies
Apricot afirma ser el primer micro popular de oficina que utiliza la nueva generación de discos flexibles pequeños. ACT eligió el microflexible de Sony, que utiliza un disco flexible de sólo 3½" de ancho, en el interior de una carcasa rígida. Esto hace que el disco sea mucho más robusto que los discos flexibles tradicionales de 5¼". Una cubierta de carga de resorte protege del polvo al microflexible



Apricot XI
La capacidad de los microflexibles es limitada, por lo que ACT ofrece el Apricot XI negro. Éste posee un disco fijo de 10 Mb que reemplaza a uno de los dos microflexibles

Aun así, da la impresión de que el Apricot no opera con la rapidez que uno esperaría de una máquina con procesador 8086.

La documentación del Apricot incluye una introducción para principiantes, una exhaustiva guía del sistema operativo MS-DOS, dos útiles guías para el Supercalc y el Superplanner y extensos manuales para el Wordstar y el Multiplan. ACT proporciona poca información en cuanto a hardware, aunque las utilidades suministradas son suficientes para instalar el ordenador. No hay ningún tipo de detalles acerca de la distribución de memoria ni de las llamadas al sistema, datos muy necesarios para una firma de software que deseara producir software independiente para el Apricot, aunque esta información se puede conseguir fácilmente de manos del fabricante.

El Apricot ha sido diseñado teniendo muy presente su utilización para gestión; no es un sistema para ingenieros de software ni para aficionados a los ordenadores. Si el Apricot alcanzara el éxito del Sirius de ACT, podemos anticipar placas conectables opcionales producidas por fabricantes independientes, así como placas de memoria extra y el modem de la propia ACT. Dejando de lado sus méritos como un ordenador de oficina sumamente versátil y económico, y prescindiendo incluso de su atrayente aspecto, el hecho de que una máquina de este precio disponga de software MS-DOS resulta suficiente para hacer que el ACT Apricot represente una opción muy atractiva.

Visualización
Un monitor de fósforo v
alta resolución proporc
visualización clara e int

LCD
Se puede utilizar una visualización en cristal líquido de dos líneas para mensajes o como reloj o calculadora

Sony Microfloppy
Se emplean microflexibles gemelos de 315 K, obteniendo un almacenamiento compacto adecuado

Teclas de función exclusiva
Estas teclas proporcionan funciones estándar como HELP y REPEAT para diversos programas

Teclas de función sensibles al tacto
Estas seis teclas se pueden etiquetar mediante la información visualizada en la pantalla de cristal líquido

RAM de 256 K
Se proporciona
RAM de 256 K



El teclado del Apricot
Además de las teclas de gran calidad que cabe esperar en todo micro de oficina, el Apricot posee seis teclas sensibles al tacto. Éstas están reservadas para funciones especiales. Debido a que estas funciones cambian de un programa a otro, el Apricot permite visualizar una etiqueta encima de cada tecla, gracias a una pantalla de cristal líquido de dos líneas de 40 caracteres, la cual también se puede utilizar para visualizar la hora



El monitor del Apricot
El monitor tiene una pantalla de sólo 9" de ancho, a pesar de lo cual tiene una visualización de textos de 132 caracteres en 50 líneas, si bien normalmente se utiliza en modalidad de 80×25 caracteres. La calidad es buena y tiene una pantalla antirreflectante incorporada. El peso del monitor reduce las posibilidades de considerar el Apricot como un portátil, aunque algunos usuarios aparentemente han optado por tener un monitor en casa y otro en la oficina, limitándose a transportar de aquí para allá el cuerpo principal del micro

Pr
de
Pa
en
se



de
na

Puerta RS232
Está disponible una conexión serial para impresoras, modems, etc.

Puerta Centronics
Conexión para una puerta Centronics estándar

ROM
La ROM del Apricot contiene un programa de arranque (*bootstrap*) y otro de autoverificación

Ranuras de ampliación
Dos ranuras de ampliación permiten disponer de placas de circuitos impresos extras, para más memoria y un modem

o estándar

dor
da-salida
ipular las funciones de
salida se utiliza un
microprocesador

Conector 8087
Hay espacio para colocar un procesador adicional 8087 para cálculos numéricos rápidos

CPU
Se trata de un microprocesador 8086 de 16 bits

Chris Stevens

ACT APRICOT

DIMENSIONES
488×413×313 mm

CPU
8086 con opción de procesador de cálculo numérico 8087

MEMORIA
256 K de RAM, ampliables a 768 K

PANTALLA
El texto se puede mostrar en 132×50 caracteres o en 80×25 caracteres. La resolución de gráficos es de 800×400 puntos, sólo en monocromático

INTERFACES
Centronics, conector para "ratón", RS232 y ranuras internas para ampliación

UNIDADES DE DISCO
Uno o dos microflexibles Sony de 315 K o 720 K de capacidad cada uno. El Apricot XI tiene un disco rígido de 10 Mb y un microflexible

SISTEMAS OPERATIVOS
MS-DOS, CP/M-86 y Concurrent CP/M-86

TECLADO
90 teclas tipo máquina de escribir más seis teclas de función sensibles al tacto, complementadas con pantalla de cristal líquido para etiquetas

DOCUMENTACION
Amplia y bien presentada

VENTAJAS
Una máquina agradable de utilizar que tiene mejores especificaciones que máquinas de oficina que son éxitos de ventas y cuestan considerablemente más. ¡Y tiene un buen aspecto!

DESVENTAJAS
A pesar de tener una buena relación calidad-precio, el Apricot es más bien caro para el aficionado. Carece, asimismo, de la capacidad para emplear el software estándar para CP/M-80

Juego de animales mágicos

Siguiendo con los programas de entretenimiento, presentamos nuestra propia versión del juego "Animales"

Animales es un juego en el cual el ordenador intenta adivinar el nombre del animal en el que está pensando el jugador. Esto lo hace formulando preguntas tales como "¿Puede volar?", "¿Es peludo?", etc. A cada pregunta, uno sólo puede responder "sí" o "no", y el ordenador gradualmente va abriéndose paso hasta un punto en el que puede aventurar una conjetura "educada". Obviamente, es bastante sorprendente, en especial para aquellas personas que no están familiarizadas con los ordenadores, que el programa sea capaz de hacer esto. Las dos características que contribuyen a que el programa sea particularmente entretenido son la capacidad del ordenador para comunicarse en lenguaje corriente (aun cuando las propias respuestas de uno se limiten a "sí" o "no") y la gran cantidad de conocimientos que necesariamente tiene que desplegar el ordenador para tener la posibilidad de adivinar de qué animal se trata.

En realidad *Animales* es un *programa heurístico* muy simple; es decir, un programa que aprende a mejorar su rendimiento durante su ejecución. Cuando el programa se utiliza por primera vez solamente "sabe" dos nombres de animales y una pregunta. Según su respuesta sea o no un "sí", puede intentar adivinar de qué animal se trata. Si el ordenador se equivoca (lo que ocurrirá casi con toda seguridad la primera vez), el programa le pide a usted que teclee el nombre del animal y una pregunta para distinguirlo de la conjetura del programa sobre cuál era ese animal. Esta información se añade entonces a la base de datos del programa para construir un "árbol del conocimiento" que pueda utilizar en el siguiente juego. Cada vez que usted juegue a *Animales*, el tamaño del árbol aumentará hasta que finalmente el programa llegue a adivinar con una precisión absoluta la mayoría de los animales, descubriendo uno nuevo tan sólo de manera ocasional.

El punto interesante a tener en cuenta es que aun así el programa no "sabe" nada acerca de animales. Sigue a ciegas una guía elaborada a partir del conocimiento combinado de todos los jugadores que han jugado con él. La información muy bien podría referirse a distintos tipos de cerveza, componentes de una motocicleta, enfermedades o a sus amigos y familiares. La versión del programa que le permite al usuario definir la pregunta inicial y dos respuestas se podría utilizar para una gran variedad de tareas diferentes. En otras palabras, no son los datos en sí mismos los que hacen que el programa funcione, sino que éste responde a la forma en que se organizan dichos datos.

Construir el árbol con un sencillo programa BASIC no es especialmente difícil. La mayoría de las estructuras como ésta se mantienen en matrices de BASIC: en este caso utilizando T$() para las preguntas y los nombres de los animales, y Y() y N() para los enlaces entre entradas concretas a T$. Estos enlaces forman el camino a través del árbol. Para cualquier entrada de T$, la correspondiente entrada en Y() le dice al programa dónde mirar si la respuesta a esa pregunta es positiva. De igual modo, la entrada en N() es el enlace para una respuesta negativa. Al final del árbol el texto en T$ no es una pregunta sino el nombre de un animal. Tanto Y() como N() están a cero en este caso y el programa tiene que hacer una conjetura acerca de qué animal en particular está pensando el jugador.

Esta versión del programa se ha hecho pequeña y simple para mostrarle los principios implicados. Si desea mejorarlo, podría perfeccionar la presentación para su máquina agregando gráficos en color, sonido, etc. Una mejora fundamental consistiría en darle al programa alguna forma para almacenar su base de datos en cinta o en disco. Las mejores versiones de *Animales* que se pueden encontrar son aquellas con las cuales unas personas han estado jugando con ellas durante años y que han construido un inmenso árbol de animales, animales míticos, objetos, personas famosas, amigos, etc., todo ello mezclado en una base de datos gigantesca.

Complementos al BASIC

Este programa está escrito en BASIC Microsoft, de modo que se lo puede ejecutar en la mayoría de las máquinas sin ninguna modificación; el único cambio que quizá se desee efectuar es en relación al formato de las instrucciones PRINT si no le gusta la visualización en pantalla.

En el Spectrum, todas las sentencias de asignación deben comenzar con la palabra clave "LET". Reescriba las siguientes líneas como se indica:

```
45  LET L=40:REM N.° max de cars. de una
      preg.
50  DIM Y(N):DIM N(N):DIM T$(N,L)
150 LET I$=A$(1):LET P$="A "
200 IF A=30 THEN PRINT:PRINT
      "ADIOS":STOP
230 IF Y(P)=0 AND N(P)=0 THEN GOTO 290
```



Árbol de aprendizaje

Este diagrama muestra el árbol de *Animales* después de haber jugado con él varias veces. Se han aprendido cinco animales diferentes, con cuatro preguntas para distinguirlos entre sí. A partir de la ejecución dada como muestra (señalada en rojo), puede verse cómo el ordenador va a utilizar el árbol para contestar a las respuestas del jugador en la próxima ocasión en que se juegue. Esta vez, el jugador está pensando en un erizo y el ordenador tiene que aprender este nuevo animal cuando descubre que el jugador no tiene en mente un león. El ordenador solicita entonces alguna forma de distinguir entre un erizo y un león, de modo que pueda aprender el nuevo animal. Una vez que tenga puesta a punto su propia versión del programa, vea si puede incluir un camello

Vive en el agua?
S **N**
Arroja agua? **Tiene franjas?**
S **N** **S** **N**
Ballena **Come árboles?** **Cebra** **León**
S **N** **Tiene púas?**
Castor **Monstruo del lago Ness** **S** **N**
Erizo **León**

Ejecucion de muestra

Hace una partida? si

Vive en el agua? no

Tiene franjas? no

El animal en el que esta pensando es un leon? no

Me rindo!!!

Cual es su animal? Erizo

Por favor, entre una pregunta que permita distinguir entre un erizo y un leon
Tiene puas?

Para leon, la respuesta seria? no

Ahora ya conozco 6 animales diferentes!

Hace una partida?

```
10  REM Juego Animales
20  REM
30  REM ** Preparación **
40  N=100: REM N.° max de animales
50  DIM Y(N), N(N), T$(N)
60  C=3:FOR I=1 TO 3:READ Y(I), N(I), T$(I):NEXT I
70  PRINT:PRINT "A N I M A L E S!":PRINT
80  GOTO 190
90  REM ** Respuesta SI o NO **
100  PRINT:PRINT Q$; "    ";:INPUT A$
110  IF A$="s" OR A$="S" OR A$="SI" OR A$="si" THEN A=1:RETURN
120  IF A$="n" OR A$="N" OR A$="NO" OR A$="no" THEN A=0:RETURN
130  PRINT:PRINT "Por favor responda SI o NO": GOTO 100
140  REM ** Añadir UN o UNA al nombre del animal **
150  I$=RIGHT$(A$,1):P$="un"
160  IF I$="A" OR I$="a" THEN P$="una"
170  A$=P$+A$:RETURN
180  REM ** Empezar un juego nuevo **
190  Q$="Hace una partida?":GOSUB 100
200  IF A=0 THEN PRINT:PRINT "ADIOS!":END
210  P=1
220  REM ** Jugar juego **
230  IF Y(P)=0 AND N(P)=0 THEN 290
240  Q$=T$(P):GOSUB 100
250  IF A=1 THEN P=Y(P)
260  IF A=0 THEN P=N(P)
270  GOTO 230
280  REM ** Hacer conjetura sobre el animal **
290  A$=T$(P):GOSUB 150:T$=A$
300  Q$="Es el animal en el que está pensando"+A$:GOSUB 100
310  IF A=1 THEN PRINT:PRINT "Lo tengo!!!!":GOTO 430
320  REM ** Aprender un animal nuevo **
330  PRINT:PRINT "Me rindo!!!":PRINT "Cual es su animal?";:INPUT N$
340  A$=N$:GOSUB 150
350  PRINT:PRINT "Por favor entre una pregunta que permita distinguir":PRINT "entre"; A$;
     "y";T$:INPUT D$
360  Q$="Para"+T$+"la respuesta seria?":GOSUB 100
370  A$=T$(P):T$(P)=D$:T$(C+1)=A$:T$(C+2)=N$
380  IF A=1 THEN Y(P)=C+1:N(P)=C+2
390  IF A=0 THEN Y(P)=C+2:N(P)=C+1
400  Y(C+1)=0:N(C+1)=0:Y(C+2)=0:N(C+2)=0
410  C=C+2
420  REM ** Final juego & bucle para otra pasada **
430  A=INT(C/2)+1
440  PRINT:PRINT "Ahora ya conozco";A; "animales diferentes!"
450  GOTO 190
460  REM ** Datos iniciales **
470  DATA 2,3, "Vive en el agua?"
480  DATA 0,0, "Ballena"
490  DATA 0,0, "Leon"
```

David Higham

Capacidades de resolución

En nuestro estudio del Commodore 64, esta vez analizaremos sus gráficos de alta y baja resolución

Para baja resolución, la pantalla del Commodore 64 se divide en 25 líneas de 40 posiciones de carácter cada una, lo cual supone 1 000 celdas en total. Como sabemos, cada carácter se construye a partir de una serie de puntos más pequeños, dispuestos en ocho filas; la celda de cada carácter consta, por consiguiente, de 64 pixels. Para conseguir alta resolución, hemos de ser capaces de encender o apagar cada pixel de forma individual utilizando un único bit de la memoria del ordenador para controlarlos uno a uno. Este concepto se conoce como *mapa de bits*. Cada posición de memoria contiene ocho bits y la pantalla 64 000 pixels, luego son necesarias 8 000 posiciones de memoria para almacenar la información de la pantalla en alta resolución.

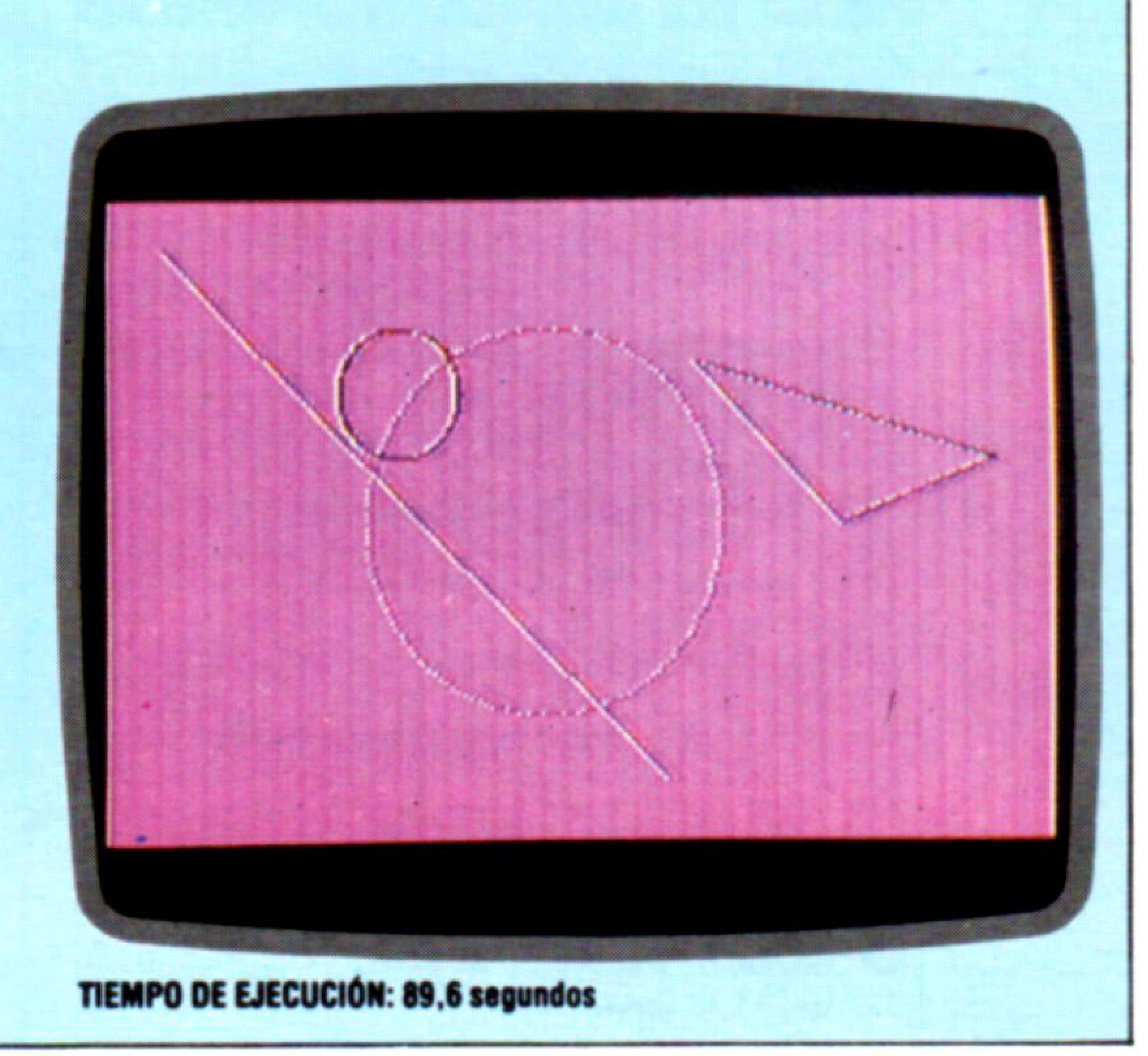
Velocidades relativas

Para producir esta visualización el Commodore 64 tardó cerca de 90 segundos con unas 50 líneas de programa. Producir la misma visualización en el Spectrum costó, sin embargo, alrededor de dos segundos mediante el siguiente programa:

```
4000 REM ******
     DEMOSTRACION ALTA
     RES *******
4050 LET N=18: DIM U(N):
     INK 6: PAPER 1: BORDER
     1: RESTORE 4100
4100 DATA 20, 160, 160, -160
4120 DATA 80, 123, 15
4140 DATA 130, 90,60
4160 DATA 175,140
4180 DATA 40, -40
4200 DATA 20,15
4220 DATA -60,25
4250 FOR K=1 TO N:READ
     U(K):NEXT K
4300 PLOT U(1), U(2):DRAW
     U(3), U(4)
4350 CIRCLE U(5), U(6), U(7)
4400 CIRCLE U(8), U(9), U(10)
4450 PLOT U(11),
     U(12):DRAW U(13),
     U(14)
4500 DRAW U(15),
     U(16):DRAW U(17),
     U(18)
4600 PAUSE 0
```



TIEMPO DE EJECUCIÓN: 1,85 segundos



TIEMPO DE EJECUCIÓN: 89,6 segundos

Liz Heaney

El Commodore 64 se pasa de la modalidad estándar de baja resolución a la de alta resolución poniendo el bit 5 de la posición 53265 a uno.

Para modificar este bit (que en decimal vale 32) sin perturbar a los otros, es necesario utilizar la siguiente instrucción:

```
POKE53265,PEEK(53265)OR32
```

Una vez establecida la modalidad de alta resolución, la pantalla recibe entonces su información desde un bloque de memoria de 8 000 bytes. El comienzo de este bloque de memoria lo señala la posición 53272. Ésta es la misma posición que se utilizó para la construcción de caracteres definidos por el usuario en el último capítulo de la serie (véase p. 712).

La zona de memoria normalmente asignada a la memoria de pantalla se emplea para contener la información de color para cada celda de ocho por ocho de la pantalla. Los 16 colores disponibles en el Commodore 64 se pueden representar con sólo cuatro bits; de modo que los cuatro bits superiores de cualquier posición de la memoria de pantalla se utilizan para indicar el color de los pixels que están "encendidos" en una celda determinada y los cuatro bits inferiores indican el color de los pixels "apagados". Por lo tanto, es posible tener pares de colores distintos, uno para el carácter y el otro para el fondo, en cada celda de la pantalla. Si deseamos un fondo púrpura, con gráficos de alta resolución dibujados en negro, son necesarios estos códigos:

El código de color para negro es 0 = 0000 en binario

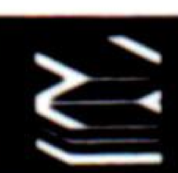
El código de color para púrpura es 4 = 0100 en binario

Juntando ambas partes obtenemos 00000100, o 4 en decimal. Colocando (POKE) 4 en cada posición de la memoria de pantalla (de 1024 a 2023) produciremos los gráficos requeridos en negro sobre un fondo púrpura.

Antes de empezar a dibujar en la pantalla en alta resolución, la zona de 8 000 bytes que controla lo que se va a ver en pantalla se debe limpiar colocando (POKE) un cero en cada posición. En BASIC esta operación llevará varios segundos. De no hacerla, la visualización en pantalla será una mescolanza de puntos, pues esa zona de memoria particular toma valores al azar cuando la máquina se conecta.

Trazando puntos

Un programa de gráficos de alta resolución ha de ser capaz de encender o apagar pixels individuales de la pantalla. Si a cada punto se le otorga una



Punto a punto
A los pixels de puntos que componen la pantalla en alta resolución del Commodore 64 no se puede acceder directamente: la pantalla para texto de 40 × 25 se asocia a un mapa de 8 000 bytes en RAM, siendo descrita cada posición de texto mediante ocho bytes. La posición de un pixel de puntos se describe en alta resolución mediante X, su distancia (en pixels) desde la izquierda de la pantalla, e Y, su distancia desde la parte superior de la pantalla. Estos números se deben convertir en la dirección del byte que contiene el pixel, y el número de bit correspondiente de ese byte

Liz Dixon

coordenada X y una Y (X e Y están comprendidas entre las escalas de 0 a 319 y de 0 a 199, respectivamente) entonces el programa puede identificar qué bit correspondiente del mapa de memoria de 8 000 bytes se ha de poner a uno o a cero.

La posición horizontal del byte se puede hallar a partir de la coordenada X con la instrucción:

```
HB = INT(X/8)
```

Del mismo modo, el correspondiente byte vertical se puede hallar a partir de la coordenada Y:

```
VB = INT(Y/8)
```

El primer byte de la celda que contiene el bit requerido, R0, se puede calcular a partir de HB y VB:

```
R0 = VB*320 + HB* 8
```

El byte que contiene el bit requerido será R0, más el resto de dividir Y por ocho. Este resto se puede hallar fácilmente a partir de los tres bits situados más a la derecha del valor de Y. Si A = YAND7 y BASE es la dirección del primer byte del bloque de 8 000 bytes, entonces la dirección del byte, BY, que contiene el bit que requerimos se puede hallar de la siguiente forma:

```
BY = BASE + R0 + A
```

El bit dentro del byte BY se puede hallar calculando el resto de dividir la coordenada X por ocho. Si B = XAND7, el siguiente POKE pondrá a uno el bit que corresponde al pixel con las coordenadas X e Y:

```
POKE BY, PEEK (BY)OR(2↑(7−B))
```

Ahora que se ha encendido individualmente cada pixel, se pueden diseñar rutinas que dibujen formas en la pantalla. El siguiente programa muestra cómo se pueden dibujar líneas rectas desde un punto (X1, Y1) a otro (X2, Y2). Un círculo se puede dibujar especificando las coordenadas de su centro (CX, CY) y el radio RA. También hay una subrutina que dibujará un triángulo a partir de las coordenadas de sus tres vértices (XA, YA), (XB, YB) y (XC, YC). Puede que a usted le guste experimentar entrando coordenadas distintas a las dadas en el programa.

Es interesante observar que la estructura de este programa se compone de un conjunto de subrutinas en serie. La rutina de nivel inferior es la que traza un único punto en la pantalla. Esta subrutina es usada por otra rutina de mayor nivel que dibuja una línea recta. A un nivel todavía superior, la rutina TRAZAR TRIANGULO utiliza la rutina TRAZAR LINEA tres veces para trazar sus tres lados. Este enfoque de programación tiene muchas ventajas. Es flexible, porque sería muy fácil diseñar una rutina para dibujar, supongamos, hexágonos regulares. Dicha rutina llamaría a la rutina TRAZAR LINEA, que a su vez llamaría a la rutina TRAZAR PUNTO. Incluso se podría utilizar la rutina DIBUJAR TRIANGULO para construir hexágonos a partir de triángulos equiláteros. En este caso la rutina DIBUJAR HEXAGONO for-

```
65 REM **** DEMOST ALTA RES ****
70 PRINT CHR$(147): REM LIMPIAR PANTALLA
80 POKE 53280,0 : REM COLOR BORDE NEGRO
90 :
100 REM **** ZONA DE MEMORIA COLOR PANTALLA ****
110 FOR I=1024 TO 2023: POKE I,4: NEXT I
120 :
130 REM **** INICIALIZAR APUNTADOR MAPA BITS ****
140 BASE=8192: POKE 53272. PEEK (53272) OR 8
150 :
160 REM **** QUITAR MODALIDAD MAPA BITS ****
170 FOR I=BASE TO BASE + 7999:POKE I,0:NEXT I
180 :
190 REM **** PONER MODALIDAD MAPA BITS ****
200 POKE 53265, PEEK(53265) OR 32
210 :
220 REM **** DIBUJAR LINEA RECTA ****
230 X1=20: X2=190: Y1=15: Y2=180
240 GOSUB 800: REM TRAZAR LINEA
250 :
300 REM **** DIBUJAR CIRCULO ****
310 CX=150: CY=100: RA=60
320 GOSUB 900: REM TRAZAR CIRCULO
330 :
370 **** OTRO CIRCULO ****
380 CX=100: CY=60: RA=20
390 GOSUB 900: TRAZAR CIRCULO
400 :
410 REM **** DIBUJAR TRIANGULO ****
420 XA=200:XB=250:XC=300: YA=50:YB=100:YC=80
430 GOSUB 600: REM TRAZAR TRIANGULO
440 :
450 GOTO 450: REM FIN DEL PROGRAMA PRINCIPAL
460 :
470 :
600 REM **** SUBRUTINA TRAZAR TRIANGULO ****
610 :
620 X1=XA: X2=XB: Y1=YA: Y2=YB
630 GOSUB 800: REM TRAZAR LINEA
640 X1=XB: X2=XC: Y1=YB: Y2=YC
650 GOSUB 800: REM TRAZAR LINEA
660 X1=XC: X2=XA: Y1=YC: Y2=YA
670 GOSUB 800: REM TRAZAR LINEA
680 RETURN
690 :
800 REM ***** SUBRUTINA TRAZAR LINEA *****
810 S=1
820 IF X2 < X1 THEN S=-1
830 FOR X=X1 TO X2 STEP S
840 Y=(Y2-Y1) * (X-X1)/(X2-X1)+Y1
850 GOSUB 1000: REM TRAZAR PUNTO
860 NEXT X
870 RETURN
880 :
900 REM **** SUBRUTINA TRAZAR CIRCULO ****
910 :
920 FOR ANGLE = 0 TO 2 * PI STEP .04
930 X=INT (RA * COS(ANGLE)+CX)
940 Y=INT (CY-RA * SIN(ANGLE))
950 GOSUB 1000: REM TRAZAR PUNTO
960 NEXT ANGLE
970 RETURN
980 :
1000 REM **** SUBRUTINA TRAZAR PUNTO ****
1010 :
1020 IF X > 319 OR X < 0 OR Y > 199 OR Y < 0 THEN GOTO 1070
1030 HB=INT(X/8): VB=INT(Y/8)
1040 RO=VB * 320 + HB*8: A= Y AND 7: B=X AND 7
1050 BY=BASE+RO+A
1060 POKE BY,PEEK (BY)OR(2^(7-B))
1070 RETURN
```

maría un cuarto eslabón en la estructura del programa.

Asegúrese de guardar (SAVE) este programa antes de ejecutarlo, ya que una instrucción POKE incorrecta haría que la máquina se "colgara" o se interrumpiera inesperadamente.

Programa Subhunter

Una parte importante del juego Subhunter que estamos diseñando es la rutina que actualiza el marcador. En un juego así hay muchas formas de asignar la puntuación; nos basaremos en estas reglas:

1) La profundidad y la velocidad del submarino son factores importantes. El que avanza con rapidez y en profundidad es más difícil de acertar que uno lento que se desplace cerca de la superficie. La puntuación asignada tendrá esto en cuenta.

2) Si se hace blanco en el submarino, el valor de su puntuación se suma al marcador del jugador, pero si el sumergible alcanza el borde de la pantalla sin haber sido alcanzado, su valor resta al marcador del jugador. No se permitirán marcadores negativos.

Más avanzado el proyecto nos ocuparemos de la rutina que selecciona al azar la velocidad y profundidad de los submarinos; pero por ahora todo lo que necesitamos saber es que la profundidad del sumergible se almacena en la variable Y3 y su velocidad en DX. El valor del submarino se puede calcular sobre esta base. Para asegurar que sólo se calculen valores de números enteros para el valor del submarino, se utiliza la función INT de la siguiente manera:

Valor sub = INT(Y3+DX*30)

Almacenaremos el valor en curso del jugador en una variable SC. Todo lo que queda por hacer es sumar o restar el valor del submarino a SC en función de si el submarino ha sido alcanzado o si se ha escapado. La subrutina ACTUALIZAR MARCADOR se utiliza en dos secciones del programa:

1) Donde se comprueba la posición del submarino para ver si ha alcanzado el borde de la pantalla y
2) durante la rutina ACERTAR

En estas dos partes del programa se puede usar el indicador DS para saber cuál de las dos partes está utilizando la subrutina ACTUALIZAR MARCADOR. Poniendo DS=1 en la rutina ACERTAR y DS=−1 en la rutina BORDE DE PANTALLA, se puede incrementar o decrementar el marcador en función del valor del submarino de la siguiente forma:

SC = SC+INT(Y3+DX*30)*DS

Luego de asegurarse de que el marcador está por debajo de cero, se puede imprimir (PRINT) el nuevo valor de SC en la línea superior de la pantalla. Añada estas líneas a su programa y guárdelo (SAVE).

```
5500 REM **** ACTUALIZAR MARCADOR ****
5510 SC=SC+INT(Y3+DX*30)*DS
5520 IFSC<0THEN SC=0
5530 PRINT<CHR$(19); CHR$(144);
     "MARCADOR";SC;CHR$(157); "    "
5540 RETURN
```

En el próximo capítulo analizaremos la creación de los sprites que se utilizarán para el barco, el submarino, las cargas de profundidad y la explosión.

El método LIFO

La pila es una zona definida de la memoria que juega un papel esencial en las subrutinas. Opera, como veremos, por el método LIFO (último en entrar, primero en salir)

El tratamiento de la memoria es la esencia de la programación en lenguaje assembly y la mayoría de las instrucciones que hemos estudiado hasta ahora en el curso están relacionadas simplemente con cargar datos en posiciones de la memoria o tomarlos de ella. A estas posiciones hemos accedido de diversas maneras (las modalidades de direccionamiento), pero hasta aquí todas las instrucciones que conocemos han empleado siempre una dirección específica de la memoria como parte del operando. Existe una clase de instrucciones, sin embargo, que acceden a un área específica de la memoria pero que no toman como operando dirección alguna. Son instrucciones que operan sobre el área de memoria denominada *pila* (*stack*) y se las conoce como operaciones de pila.

La pila está creada para que tanto la CPU como el programador dispongan de una memoria temporal donde trabajar. Se trata de una especie de "cuaderno de apuntes" cómodo para escribir y también fácil de leer y borrar. Las operaciones de pila copian datos de los registros de la CPU en zonas libres de la pila, o copian datos de la pila en los registros de la CPU. Estas instrucciones no exigen un operando con dirección, ya que un registro específico de la CPU, el llamado *índice de pila* (*stack pointer*), siempre contiene la dirección de la primera posición libre de la pila. Por tanto, todo cuanto se escribe en la pila es depositado automáticamente en el byte señalado por dicho índice, y los bytes que se sacan de la pila se toman de la última posición utilizada. Al ejecutar una operación de pila, su índice se ajusta como parte de la operación.

En los sistemas 6502, la pila ocupa los 256 bytes de RAM que van del $0100 al $01FF; en los sistemas Z80, la situación y el tamaño de la pila son determinados por el sistema operativo, pero pueden ser modificados por el programador. Esta variación refleja las diferencias de los microprocesadores en cuanto a organización interna (véase el diagrama de la página 616): el 6502 posee un índice de pila de un solo byte, mientras que el del Z80 se compone de dos bytes.

La CPU trata el contenido del índice de pila del 6502 como el byte bajo de una dirección de pila, al cual, para completar la dirección, se le suma automáticamente un byte alto con valor $01. Este bit extra siempre está a 1, de modo que en el 6502 todas las direcciones de pila se hallan en la p. 1.

El índice de pila del Z80 es un registro de dos bytes capaz de direccionar cualquier posición comprendida entre $0000 y $FFFF, o sea, todo el espacio direccionable por el propio Z80. Por consiguiente, la pila puede estar situada en cualquier lugar de la RAM, y el programador puede cambiarla de sitio si lo desea. Sin embargo, esto no es aconsejable, dado que el sistema operativo establece inicialmente su situación y va almacenando datos en la misma. Ya que el sistema operativo puede interrumpir la ejecución de cualquier programa en lenguaje máquina en cualquier momento, esperando hallar en la pila los datos adecuados para su funcionamiento, cualquier alteración de la situación de la pila significaría el desconcierto del sistema operativo, que puede quedar colgado a causa de esto.

Un ejemplo de cómo se emplea la pila será la siguiente rutina para intercambiar el contenido de dos posiciones de memoria, LOC1 y LOC2:

6502		Z80	
LDA	LOC1	LD	A,(LOC1)
PHA		PUSH	AF
LDA	LOC2	LD	A,(LOC2)
STA	LOC1	LD	(LOC1),A
PLA		POP	AF
STA	LOC2	LD	(LOC2), A

Primera línea: lo que contiene LOC1 es cargado en el acumulador (LDA). Segunda línea: el acumulador es "empujado" (*push*) dentro de la pila (PHA). Tercera y cuarta líneas: el contenido de LOC2 se carga en el acumulador y seguidamente se almacena (STA) en LOC1. Penúltima línea: el último (*Last*) byte metido en la pila "salta" (*pops*) otra vez dentro del acumulador (PLA). Última línea: el actual contenido del acumulador (que antes estaba en la pila y al principio en LOC1) se almacena en LOC2. Fin del intercambio. Observe cómo las operaciones de pila (PHA, PLA, o bien PUSH y POP AF) no llevan indicación de dirección alguna, a no ser, implícitamente, la primera posición libre en la pila.

Este fragmento de programa nos muestra muchísimas cosas acerca de las operaciones de pila. Ante todo, que son recíprocas y secuenciales. El último ítem empujado dentro de la pila se recupera al primer salto que se ordene desde la pila. Si ha habido varios "empujones" seguidos sin ningún "salto", los datos se escriben en sucesivas posiciones de la pila, cada byte "encima" del anterior, y si se piden varios "saltos" consecutivos, éstos van afectando a las sucesivas posiciones en orden descendente.

Una imagen visual de la pila la tendría si fuera escribiendo durante unos días numerosas tarjetas postales y las fuera apilando en un cajón. La que ocupa en cualquier momento la parte superior del montón sería la más reciente, la del fondo del montón la más antigua, y para sacar ésta debería sacar todas las anteriores. Por esta razón se dice que la pila es una estructura LIFO (*Last In First Out*), o sea, el último que entró es el primero en salir. Su

opuesta, la estructura FIFO (*First In First Out*), o sea, el primero que entra es el primero que sale, es denominada "cola" en lugar de pila. Convencionalmente, al primer byte libre de la pila se le denomina *tope* (*top*, en inglés) de la pila, ya que uno se imagina las pilas creciendo hacia arriba. Aunque, de hecho, tanto en el 6502 como en el Z80, el índice de pila disminuye a cada empujón, por lo que el tope está en realidad en una posición de memoria inferior a la base de la pila.

El primer fragmento de programa también constituye un ejemplo típico de los programas que utilizan la pila, en cuanto a que el número de instrucciones de empujón está exactamente equilibrado con el número de saltos. Esto no es esencial, pero el no observar este equilibrio de opuestos al escribir subrutinas podría acabar en un retorno incorrecto desde la subrutina y en el consiguiente fallo del programa. Este error se puede rastrear comparando el número de instrucciones de saltos y empujones del programa.

La versión Z80 de la rutina difiere notablemente de la del 6502 en un aspecto fundamental: el 6502 empuja en la pila un registro de un solo byte, mientras que el Z80 siempre empuja un registro de dos bytes. Cuando usted empuja o hace saltar el acumulador del Z80, también empuja o hace saltar el registro de estado del procesador, porque la CPU trata estos dos registros de byte único como un registro de dos bytes denominado AF (*Accumulator Flag*: acumulador-indicador).

Es una buena costumbre del programador comenzar las rutinas poniendo los contenidos de todos los registros de la CPU en la pila y sacándolos de la pila inmediatamente antes del retorno de la subrutina. Esto asegura que la CPU, después de la llamada a la subrutina, se mantenga exactamente en el mismo estado en que se encontraba antes de la misma, y significa que en la subrutina se puede utilizar cualquiera de los registros sin temor de corromper datos esenciales para el programa principal. Por ejemplo, consideremos esta subrutina:

	6502		Z80	
	LDA	LOC1	LD	A,LOC1
SUM	ADC	#$6C	ADC	A,$6C
GSUB	JSR	SUBR0	CALL	SUBR0
TEST	BNE	SUM	JR	NZ,SUM
EXIT	RTS		RET	
SUBR0	PHP		PUSH	AF
	PHA		PUSH	HL
	TXA		PUSH	DE
	PHA		PUSH	BC
	TYA		PUSH	IX
SUBR1	PHA		PUSH	IY
SUBR2	STA	LOC2	LD	(LOC2),A
	LDA	#$00	LD	A,$00
SUBR3	PLA		POP	IY
	TAY		POP	IX
	PLA		POP	DE
	TAX		POP	BC
	PLA		POP	HL
SUBR4	PLP		POP	AF
	RTS		RET	

Aquí el efecto de las instrucciones entre SUBR0 y SUBR1 es el de empujar hasta la pila los contenidos

ANTES
CONTADOR DEL PROGRAMA
E946
$0100
$015F PRÓX BYTE LIBRE
ÍNDICE DE PILA
$01FF
PILA
INSTRUCCIÓN
JSR $F000
JSR $E943
00 $E944
F0 $E945
CLC $E946
RTS $F000
MEMORIA DEL PROGRAMA
DESPUÉS
CONTADOR DEL PROGRAMA
F000
$0100
$015D PRÓX BYTE LIBRE
$015E 46
$015F E9
ÍNDICE DE PILA
$01FF
PILA

La pila

La pila es esencial en el tratamiento de llamadas a subrutinas (como la JSR del 6502). Cuando se carga por primera vez la instrucción "saltar a subrutina", el contador del programa contiene la dirección de la siguiente instrucción del programa ($E946 en este caso, la dirección de la instrucción CLC); esta dirección se "empuja" hasta la pila haciendo que el índice de pila pase a apuntar al siguiente byte libre de la pila ($015D, en este caso), y que el operando de la instrucción de salto ($F000 aquí) se cargue en el contador del programa como la dirección de la instrucción siguiente, causando, por consiguiente, una bifurcación en el flujo del programa

Kevin Jones

actuales del registro, y el efecto de las instrucciones entre SUBR3 y SUBR4 es el de restaurar de nuevo aquellos contenidos en los registros. Las instrucciones substanciales de la subrutina son las dos que empiezan en SUBR2, pero la segunda de éstas es ineficaz, dado que las instrucciones siguientes cambian completamente el estado del acumulador.

Observe que las instrucciones PUSH y POP del Z80 pueden tomar como operando cualquier par de registros, pero el 6502 sólo puede operar sobre el acumulador (PHA y PLA) y sobre el registro indicador de estado (PHP y PLP). De ahí la necesidad de las transferencias registro-acumulador (TXA, TAX, TYA, TAY) de la versión 6502. Observe también que en la versión Z80 hemos cometido un error deliberado al no sacar todos los registros por el orden inverso al que fueron colocados. Ello ilustra el cuidado que es necesario tener en las operaciones de pila, pero demuestra, asimismo, que usted puede empujar a pila un registro y después hacer saltar ese valor hasta un registro diferente: una forma laboriosa pero a veces convenientes de realizar transferencias de datos entre registros.

Las funciones y los usos de los registros de la CPU será tema del próximo capítulo, con el cual terminaremos nuestro análisis general de las instrucciones del lenguaje assembly. Iniciaremos además el estudio de la aritmética en lenguaje máquina.

Ejercicios

1) Reescribir la segunda rutina que empleamos en las soluciones de los ejercicios anteriores, de tal forma que el mensaje de LABL1 se vuelva a almacenar en LABL1, pero en orden inverso, así:

LABL1 EJASNEM NU SE OTSE

Para esta inversión utilice la pila.

2) Desarrollar esta rutina de modo que las palabras del mensaje permanezcan en el orden original, pero que se inviertan los caracteres de cada palabra:

LABL1 OTSE SE NU EJASNEM

Respuestas a los ejercicios de p. 717

1) Esta subrutina almacena los números de $0F a $00 en orden descendente en el bloque de $10 bytes reservado en LABL1 por el pseudo-op DS.

6502		Z80	
ORIGIN	ORG $7000	ORIGIN	ORG $C000
LABL1	DS $10	LABL1	DS $10
LABL2	DW $7100	LABL2	DW $C100
		OFFST	EQU $0F
BEGIN	LDY #$FF	BEGIN	LD IX,LABL1
	LDX #$10		LD B,OFFST
LOOP0	INY	LOOP0	LD (IX+0),B
	DEX		INC IX
	TXA	ENDLP0	DJNZ LOOP0
	STA LABL1,Y		LD (IX+0),B
ENDLP0	BNE LOOP0		RET
	RTS		

Las diferencias en cuanto a enfoque e instrucciones entre el Z80 y el 6502 son reveladoras. El 6502 utiliza el registro Y como un índice a la dirección LABL1, y el registro X como un contador de bucle y fuente de los datos a almacenar. Observe que el registro X es decrementado dos instrucciones antes de la comparación BNE e ENDLP0, pero como TXA (Transferir el contenido de X al acumulador) y STA no afectan al registro indicador de estado, la comparación actúa sobre los efectos de decrementar X.

La versión Z80 utiliza la modalidad de direccionamiento indirecto IX para retener la dirección de almacenamiento, y utiliza el registro B como contador y fuente de datos. En ENDLP0, en la versión Z80, vemos DJNZ LOOP0, que significa "decrementar el registro B y saltar a LOOP0 si el resultado no es cero". Esta instrucción es casi una estructura FOR...NEXT del lenguaje assembly.

2) Esta rutina copia el mensaje almacenado en LABL1 sobre el bloque que comienza en la dirección almacenada en LABL2. El valor $0D (el código ASCII para Return o Enter) se almacena al final del mensaje como terminador.

6502		Z80	
ORIGIN	ORG $7000	ORIGIN	ORG $C000
LABL1	DB 'THIS IS A MESSAGE'	LABL1	DB 'THIS IS A MESSAGE'
TERMN8	DB $0D	TERMN8	DB $0D
LABL2	DW $7100	LABL2	DW $C100
CR	EQU $0D	CR	EQU $0D
ZPLO	EQU $FB		
BEGIN	LDA LABL2	BEGIN	LD IX,LABL1
	STA ZPLO		LD IY,(LABL2)
	LDA LABL2+1	LOOP0	LD A,(IX+0)
	STA ZPLO+1		LD (IY+0),A
	LDY $FF		INC IX
LOOP0	INY		INC IY
	LDA LABL1,Y		CP CR
	STA (ZPLO),Y	ENDLP0	JR NZ,LOOP0
	CMP CR		RET
ENDLP0	BNE LOOP0		
	RTS		

La versión 6502 utiliza al registro Y como un índice para la dirección indirecta ZPLO, en modo indirecto postindexado. Esta modalidad sólo es posible con el registro Y, y exige una dirección de operando de página cero; de ahí la inicialización de ZPLO y ZPLO + 1 con la dirección almacenada en LABL2. El sistema operativo de las máquinas 6502 emplea la mayoría de las posiciones de página cero; pero en el Commodore 64 las posiciones desde $FB hasta $FF no se utilizan, así como tampoco se usan las posiciones desde $70 hasta $8F en el BBC Micro, de modo que ZPLO se pone en una de estas posiciones. La versión Z80 emplea IX en modo indexado y IY en modo indirecto indexado.

Ambas rutinas utilizan una instrucción "comparar el acumulador" —CMP CR (6502) y CP CR (Z80)—, en la cual al contenido de éste se le resta el operando, afectando a los flags del indicador de estado (PSR). El contenido del acumulador se restaura luego, mientras el PSR muestra el resultado de la comparación. Cuando el acumulador contenga $0D (fin del mensaje), el resultado será poner a 1 el flag cero. De modo que la comparación ENDLP0 fallará y el control pasará a la instrucción de retorno.

Cambridge Connection

Camputers es otra empresa informática de primera línea, instalada en Cambridge, el "Silicon Valley" británico


John Shirreff

Camputers nació de la inspiración de un solo hombre. En 1976, Dick Greenwood empezó a trabajar como diseñador independiente de electrónica, aceptando contratos de firmas tales como Pye Telecoms. Hacia finales de los años setenta, Greenwood había creado su propia empresa y se había iniciado en un trabajo de desarrollo más especializado, también basado en contratos. La empresa acabó abarcando la creación de software, produciendo un Bar Management System, paquete para control de existencias que permitía a las fábricas de cerveza controlar sus ventas a bares y otros establecimientos.

La empresa pasó luego al diseño de microordenadores, concentrándose en proyectos basados en el chip Z80 de Zilog. En febrero de 1981, Greenwood creó Camtronic Circuits (que luego sería Camputers) y con una subvención estatal empezó a trabajar en el ordenador personal Lynx en el verano de 1981. El deseo explícito de Greenwood era el de "enseñarle al Z80A a bailar en torno a los problemas, no a arrastrarse a través de ellos".

En cuanto a la parte de la creación del hardware del proyecto, quien estaba a su cargo era John Shirreff, graduado por la Universidad de Cambridge, que se incorporó a la empresa después de trabajar en la industria de la música rock. La elaboración del software la llevaba Davis Jansons, quien escribió la versión de BASIC utilizada por la máquina.


Sistema para gestión
En la fotografía vemos el sistema modular Lynx Laureate diseñado para usuarios de gestión. Los 128 K de memoria incorporados le permiten aceptar CP/M

Cortesía de Camputers

Cuando apareció, en 1982, el Lynx se consideró una máquina de apariencia muy profesional. Empaquetado en una atractiva carcasa gris, con un teclado completo tipo máquina de escribir y disposición QWERTY, el ordenador venía equipado con una memoria estándar de 48 Kbytes que se podía ampliar a 192 Kbytes. La máquina tenía capacidad para visualizar hasta ocho colores diferentes, con una modalidad de alta resolución de 248 × 256 pixels. También tenía un altavoz incorporado para aprovechar al máximo sus capacidades de audio.

A decir verdad, el Lynx nunca alcanzó gran popularidad en Gran Bretaña, si bien las ventas en el extranjero alentaron a la empresa a seguir desarrollando el diseño básico. Al cabo de poco tiempo apareció el Lynx 96, que venía equipado con cerca de 37 Kbytes de RAM para el usuario, así como con la capacidad de soportar unidades de disco flexible de 5¼". Además, el Lynx 96 venía con efectos de sonido preelaborados. La máquina disponía, como opciones adicionales, de interfaces para impresora en serie y en paralelo.

Más recientemente, la empresa ha introducido una máquina destinada al sector de pequeña gestión del mercado del microordenador. El ordenador se denomina Lynx Laureate y se basa en el microprocesador Z80, puede operar bajo el sistema operativo CP/M, que proporciona al usuario acceso a la enorme cantidad de software CP/M que se ha escrito en el transcurso de la última década. A pesar de haber sido diseñado como una máquina de oficina pequeña, el Laureate es, sin embargo, compatible con los otros modelos Lynx y está dotado de un bus de ampliación de 40 vías que le permite utilizar toda la gama de paquetes para periféricos Lynx, incluyendo una impresora en paralelo, palanca de mando y software basado en ROM.

Camputers, dirigida por su actual presidente Stanley Charles, continúa desarrollando productos nuevos. En un futuro cercano, la empresa tiene planeado lanzar una versión alternativa de la máquina de oficina Laureate. Este sistema se podrá adquirir como un paquete integrado. También hay planes para relanzar el Lynx 48 en Gran Bretaña, con el nombre de Leisure. Éste estará dirigido específicamente al mercado del ordenador personal y para juegos, sector en el que la empresa considera que sus productos han sido injustamente ignorados. Con los ojos puestos en un futuro algo más lejano, Camputers está trabajando en una máquina que la empresa espera que se convierta en el competidor directo del Sinclair QL.


mi computer
Editorial Delta, S.A.


E R T Y U I O P
D F G H J K L
X C V B N M

10037
9 788485 822836